IMAGE PROCESSING WORKBENCH: STATISTICAL, MATHEMATICAL AND LOGICAL TOOLS

by

Major Narinder Singh

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

APRIL, 1995

IMAGE PROCESSING WORKBENCH: STATISTICAL, MATHEMATICAL AND LOGICAL TOOLS.

A thesis submitted

in partial fulfillment of the requirements

for the degree of

Master of Technology

by

Major Narinder Singh

to the

Department of Computer Science & Engineering

Indian Institute of Technology , Kanpur

CERTIFICATE

It is certified that the work outlined in this thesis titled **Image Processing Workbench: Statistical, Mathematical and Logical Tools** by Major Narinder Singh has been carried out under my supervision and that this work has not been submitted elsewhere for a degree

[illegible]

April 1995
Kanpur

Dr. H Karnick
Assosciate professor
Dept. of Comp. Sc. and Engg.
I.I.T., Kanpur

ABSTRACT

This thesis implements part of an image processing work bench and incorporates statistical, mathematical and logical operations on images A companion thesis has implemented the filtering and morphological operations.

The work bench is implemented on top of MS Windows and provides a flexible graphically oriented environment for processing images.

ACKNOWLEDGMENTS

I would take this opportunity to express my sincere gratitude towards Dr H Karnick, my thesis supervisor for having lit in my heart the flame symbolizing the spirit of enquiry, for having encouraged me to make an attempt against all odds and for standing by me in my moments of sorrow and joy. His regular support was quite instrumental for having laid threadbare the mystique and mysteries of *Image Processing*. Often during the course of thesis work I had to seek his elderly advice in sorting my personal problems, for which he gave a very patient hearing I thank my mother, my wife and my children for their staunch support. I would cherish my association which I had forged here with Vidyut Varma, Major AK Nath and everybody else in CS stream who were a great support during my thesis. I would be failing in my duty if I do not express my gratitude towards *Corps of Signals* who have afforded me this great opportunity.

Narinder Singh

Contents

List of Figures

Chapter 1

Introduction

1.1 Image Processing and its Applications

Images are two-dimensional representation of the visual world. They are encountered in a wide variety of disciplines including art, human vision, astronomy and engineering. *Image Processing* is a term used to describe operations carried out on images, with the aim of accomplishing some purpose. This might be, for example, to convert the image into a form where it can be more easily transmitted over a telecommunication link or stored in computer memory ; it might also be to reduce noise or to extract information of particular interest to a human observer. Common image processing operations include histogramming, profiling and filtering among many others.

Several methods, including optical ones, can be used to process images, but most image processing is accomplished electronically, where it is possible to use the power and sophistication of modern computing hardware and software. As computers become faster, cheaper and smaller with larger memories, processing images without using extra special hardware becomes possible.

Interest in image processing methods stems from two principal application areas: improvement of pictorial information for human interpretation, and processing of scene data for autonomous machine perception. One of the first applications of image processing techniques was in improving digitized newspaper pictures sent by submarine cable between London and New York. Image Processing is used extensively in commercial art applications involving the retouching and rearranging of sections of photographs and other artwork. Medical applications make use of image processing techniques both for picture enhancements and in tomography. Other medical imaging techniques include ultrasonics and nuclear medicine scanners . Numerous other fields make use of imaging techniques to generate pictures and analyze collected data. Satellite photos are used to analyze terrain features. Data from solar flares can be plotted and galaxies are reconstructed from astronomical observations collected and arranged in large databases.

1.2 Aim and Scope

The primary goal of this thesis is to build a portable PC based library of image processing routines which can be used in a flexible manner through an easy to use graphical user interface These library routines can be used for image processing applications as part of a bigger system. The routines written have been so that they will be compatible with different video hardware and MS Windows software.

1.3 Adaptability for use as work bench

The utility of any software package is gauged not just by the functionality that it provides or for that matter how efficiently it has been implemented. It should be possible to use it as a means for quickly creating useful applications prototypes. Unless a user is able to quickly customize the utility's behavior pattern provided by the software package in a particular predetermined sequence, rapid and quick results may not be attainable.

As a consequence of this and other factors, much of the opportunity as well as the large body of knowledge remains inaccessible to application-oriented researchers from within the image processing community and from areas such as computer graphics, robotics and pattern recognition which has been source of many image processing problems.

The work bench implemented as part of this thesis has been designed to incorporate the above requirements. Its *Batch Programming Mode* allows a user to compose image processing operations which helps in quickly prototyping applications.

1.4 Brief over view of work done

The work bench apart from the user interface, includes collection of routines written for arithmetic, boolean and geometric operations. It also includes routines for performing statistical and grayscaling operations followed by various thresholding techniques The user is at liberty to modify the default attributes such as Pen for making graphical plots, Brush for filling the background and Font for customized labeling of the results. This is supported by a set of input and output routines for loading, displaying, saving and printing of images It enables a user to operate on a single input image or two input images. User can process a portion of the image, in case of a single input image. In keeping with the requirements of a work bench, this package enables a user to program in batch mode to quickly perform certain operations in obtaining the desired results. This package also enables a user to have access to the context sensitive help that can be obtained on the current topic through the help menu provided with it.

1.5 Overview of thesis

Chapter 2 gives the Overall design of the package followed by Chapter 3 which acquaints with the menu items and choices that will be encountered as the package is run. Chapter 4 gives a brief introduction of the algorithms being used. Chapter 5 gives the results obtained after processing few images. Chapter 6 concludes the thesis giving few suggestions for further improvements.

Appendix A at the end gives the flow chart of the general sequence of action that is to be followed for an operation. Appendix B gives definition of few terms that are used in the thesis report. Appendix C discusses the bitmap format followed by Appendix D which includes a *User Manual* for using this work bench.

Chapter 2

Overall Design Of The Package

This chapter discusses the overall design of the package and give reasons for the design choices made.

2.1 MS Windows Platform

Before a shift was made towards MS Windows, work for this project was first begun in MS DOS. Primary issue that was to be addressed was, how to obtain a faithful display of the original image on the monitor. In fact this was the central issue all through that figured in dictating the course of thesis work which was to fall in favor of MS Windows later due to certain problems that was noted when working in MS DOS.

When work was begun in MS DOS, recourse to two specific approaches was taken, in an attempt to display images on screen. Both being termed as direct approaches, namely - ROM BIOS routines and direct access to video memory. These techniques were found to be slow and not quite flexible. Speed and Results of these techniques depended to a greater extent on the Video adapter being used, video RAM supported by it, monitor being used and the underlying CPU on which the whole game is being played. Following notable flaws were found with these direct techniques:

- ROM BIOS routines operated pixel by pixel, hence took considerable amount of time to display an image.
- Direct Video memory access, although, comparatively faster, could not be easily ported as it required program to calculate the memory address of the pixel location besides being aware of the pattern in which the even and odd rows are arranged in the video memory borne by the video

adapter (a feature which is specific to video adapters and is strictly hardware dependent).

- Aspect Ratio of the monitor screen produced unwarranted distortions in the displayed image. To remedy this anomaly an attempt was made using C run time library functions to correct the aspect ratio problem. It did not work as it was learnt that that C run time routines are effective for those images/pictures which were constructed using C run time functions alone
- Jaggies, a stair step effect was noticed on diagonal lines and edges.
- Unfaithful display, this effect was more pronounced if the image had more graylevels than what the video hardware can display. In particular taking example of 16 color VGA adapter in its video mode 18 (dec) has a screen resolution of 640*480. It will pose a problem if an image with more than 16 graylevels are displayed. Using video mode 19 (dec) which of course gives full complement of 256 colors but distorts the displayable image, at the reduced resolution that it offers. In this mode this adapter gives screen resolution of 320*200.
- The greater the resolution of a graphics system, and the more colors, the more memory is required on the graphics adapter. Taking example of standard VGA graphics adapter, its minimum video RAM is 256KB, which means it is possible to display any image size up to this limit directly on the screen. If image size to be displayed is greater than this size then either additional video RAM is added to the video adapter or some sort of banding has to be implemented to send output to the monitor through the video adapter. The point in issue is the greatest image size that can be displayed by the video adapter with its original capabilities. In real life image sizes can be more than 256KB. A 256 graylevels image with full screen resolution of 640*480, makes up 300KB alone. VGA adapters with more video RAM can provide solution

to present displayable limit in DOS based programs if banding is not to be implemented.

C run time graphic functions provided alternative to the above direct means . In that they had the advantage of being portable and convenient to use. They, however, had the disadvantage of being slower than the direct means because at the core level these routines call ROM BIOS routines for display. No significant improvements were seen in the results .

Even if the above techniques were accepted, for displaying the image, with all its flaws it would have entailed considerable effort, from programming point of view. For images larger than the size of monitor screens, would have required implementation of banding and scrolling to have a complete view of the of the image. MS Windows with all its advantages in terms of displaying, scrolling, menus and still with many more advantages as will be seen becomes the favorable candidate by providing the necessary platform for thesis work leading to the rejection of further work in MS DOS environment.

As we had just seen various flaws and disadvantages of working in MS DOS environment. MS Windows was found to be suitable as it not only overcame various shortcomings seen in MS DOS environment but also provided its overly rich set of GDI (Graphics Device Interface) routines for handling bitmaps and other routines for construction of dialog boxes and menus MS Windows extends the power of MS DOS by letting one machine run several programs at the same time, all with an easy, standard, point and click interface. *Multitasking*, *device independence*, and a *standard graphical interface* are other features that distinguishes MS Windows from other programming environments.

2.2 Image Display

As soon as the image is selected by the user through the **open dialog box,** following actions are taken by the read procedure:

- Open the image file.
- Read the **BITMAPFILEHEADER** structure.

- Verify the header contents. The value in the **bmfType** field should be "BM" (meaning its an bitmap image).
- Determine the size in bytes of the data we have to load.
- Allocate that much room in global memory and lock it (locking is necessary as Windows might move the block while doing its usual memory management resulting in change of memory address).
- Read the bits from the file into the new global memory object.
- Verify that Windows bitmap is loaded and not OS/2 bitmap (which is very similar).
- Destroy the previous bitmap, if any, and set the new one in its place.
- Create new palette out of the DIB (Device Independent Bitmap).

Paint procedure is called which first creates a DDB (Device Dependent Bitmap) from the DIB, realizes its palette before drawing on the memory bitmap and then makes a final blit to the screen.

The display routine can read 2, 16, or 256 graylevel images including images that are run length encoded (RLE). Before processing, program converts images that do not have 256 graylevels, to an image with 256 graylevels. Pixel graylevels are scaled to the range 0 - 255, after processing is over. Images should be in the native Windows bitmap format referenced with the extension .BMP. The format specifications are given at appendix C.

To facilitate quicker display response and realize true colors of the DIB. The color table of the DIB is realized as a logical palette in the current DC and then the bitmap is created based on this DC. The DDB can now be used for display using GDI routines. By converting a DIB to DDB not only colors of the original DIB are preserved but also the display process is speeded up as the nearest-color search would have to be performed only once, and then the bitmap object would be in the same format as the video display.

The user can save the entire processed file or part of it. In case the entire file is saved the header information is unchanged except for the matrix multiplication operation when the output image may have changed dimensions. Rest of the pixel data is available in a buffer which can be saved right away. In case part of the image is saved, both the header and pointer to pixel data are modified before writing to disk.

2.2.1 DIB and DDB

Device Dependent Bitmaps (DDB) are simple bitmaps without any color information as part of their structure. They have to depend on the DC's default palette for their bit of color information and thus must be compatible with a specific graphics output device On the contrary DIB format is called device independent because it contains a color table. The color table describes how the pixel values correspond to RGB values. This color table may not necessarily be compatible with a particular graphics output device. A DIB is called a packed DIB if its pixel data begins immediately following the **BITMAPINFO** header, which usually is the case. Unlike DDB, which is a GDI object and can be selected into the DC, DIB cannot be selected into the DC.

They have their relative advantages and disadvantages. DIB's are primarily for interchange among programs. They can be passed among programs either by storing them in files or copying them to the clipboard. DDB's can speedily be displayed so it is important DIB's be converted to DDB's for displaying.

2.3 User tools

This package allows an user to perform a variety of operations starting with statistical, grayscaling, thresholding and cut-paste operations to name a few. Most of these functions have been linked together to form a DLL (dynamic link library) to allow different applications to call the same functions at run time. The details about each function implemented are covered in the subsequent chapters. Other features provided with the package are:

- Option to work on single or two input image files. In case of single input image option, User can further specify the portion that is to be processed.
- Scroll bar logic to deal with large files.

- View output available in various buffers (plots and processed images) or the original unprocessed image at any time.
- The whole image or a part of it can be saved to disk.
- Printing the DIB files and exchanging images from within the program itself using the clipboard.
- Ability to compress (RLE) and uncompress images.
- Batch Programming for batch mode operations. Batch default values dialog box enables viewing/editing of default settings.
- Ability to specify color, size, style of the Pen, Brush and Font required for Plots, Background and Labeling respectively.
- Use of custom controls such as Borland Windows Custom Controls (BWCC) rather than Windows resources for construction of dialog boxes and message boxes.
- Provision of context sensitive help.

2.4 Software Used

This package has been compiled and linked in large memory model using Borland C++ Version 4.0 in the IDE (Integrated Development Environment) on a 486 CPU based PC. This package can be run on MS Windows Version 3.1 and above. Due consideration has been given to portability and compatibility issues to ensure that this package can be used with later versions of MS Windows (such as Win32 etc.) with little or no modifications. Following precautions, thus, were taken:

- ♦ All modules of the package have been compiled using the preprocessor directive **STRICT**. Defining STRICT at the beginning of the module turns on some extra parts of **windows.h** (main Windows header file) that enable the compiler to check function arguments and return types more carefully than it otherwise can.
- ♦ Use of extensive macros provided with **windowx.h** (its Windows 3.1, supplementary header file). This header file defines, among other things,

additional API functions in the form of macros that call other functions. These macros not only produces better code but will be also compatible with Win32.

Chapter 3

A Tour of the Work bench

3.1 Components of Package

This chapter provides a tour through the various facilities in the work bench package.

3.1.1 Arithmetic Operations

These routines can be used for various arithmetical operations like addition, subtraction, multiplication and division. In case of multiplication routines provision for dot and matrix multiplication's are present. These operations can be used with a single input image or two input images. In case of single input image, the user can specify the portion of the input file to be processed. These elementary arithmetic operations are not performed in isolation but actually performed as part of other more complex operations. Results after the operation are displayed in the current window. Apart from the processed results the original input image or images (in case of two input images) are also displayed.

3.1.2 Boolean Operations

These are routines that perform various Boolean operations on input images. They include routines for Boolean NOT, Boolean OR, Boolean NOR, Boolean AND, Boolean NAND, and Boolean XOR. With the exception of Boolean NOT operation, all Boolean operations take two images as input. Results are displayed in a similar manner on the screen as for Arithmetic operations, with separate buffers for input and output images that can be viewed separately on the screen.

3.1.3 Geometric Operations

Geometric Operations contain set of basic routines that can be used for performing various geometric operations on input images. They include routines for cutting and pasting of images and other graphical drawings, routines for flipping and zooming. Images can be flipped horizontally, vertically, or both ways simultaneously. Geometric operations also include routines for drawing geometrical shapes such as ellipses, rectangles, lines etc. interactively. These geometrical shapes can be superimposed on top of the bitmap images and operations can be done on the resultant image. It also enables a user to choose from a selection of different pens for drawing the geometrical shapes, brush for filling the interior and the background. This can be combined with the Raster Operation Codes (ROP) to have the desired effect on the output. A clipboarding feature has been implemented as part of the package. Bitmap images can be exported or imported. It also enables a user to view the specifications of a bitmap image through a dialog box. Run length encoded images can be read and images can be written out in RLE format. Images can be rotated by a specified amount in either direction (clockwise or anti clockwise).

3.1.4 Statistical, Grayscaling and Thresholding Operations

Statistical operations like histogramming and profiling are available. As in other single input image operations, these operations can be performed on part of the image. Histogramming, will give a pictorial representation of the histogram of the input image. Profiling enables a user to perform profile analysis of the selected portion of the image in both horizontal or vertical directions.

Grayscaling operations include routines for linear and non linear grayscaling, autoscaling, equalization and histogram specification. For linear and non linear grayscaling an user can specify the input values through the dialog box that follows. Similarly, the range of histogram values desired through the dialog box for autoscaling can be specified. In histogram specification a user is required to specify the desired histogram through the dialog box. Since 256 values are required for a complete specification to specify whether the final output should be bright or dark. Accordingly, the program computes the desired histogram before proceeding with rest of the operation. In case a user does not input all the 256 values and aborts prematurely, the program goes ahead with the desired histogram which yields a brighter output. Grayscaling operations can be performed on a single input image.

Thresholding operation enables a user in converting a multigraylevel image to a binary image (image with only two levels). Three methods for deciding the threshold values are supported namely - User Defined, by Edge Detection and Dynamically to binarize an input image.

In grayscaling and thresholding operations, user can view the processed image and its histogram on the screen. Also the program computes the histogram of the original image which can be separately viewed. In case of linear and non linear gray scaling operations instead of the histogram for the processed image the plot of linear or non linear transformation function is displayed For profile operations, pictorial representation of the chosen profile direction (horizontal or vertical) is plotted in the current window.

3.1.5 Attributes and Input/Output Operations

Presence of a color monitor is of paramount importance to see the real effect of various attributes that come into play. They include style, size, and color of the pen. An user can specify it before any operation that yields plots (statistical, grayscaling and thresholding operations) as one of its outputs. The plot will bear the characteristics of the last pen selected. A brush enables a user to choose from solid or hatch patterns, in multiple styles or colors of it. A brush paints the background screen as the output is being displayed. Font enables a user to choose a font, decide its color, style and size. It can be used in labeling the output on screen.

In input/output (I/O) routines, a user can load/store an input and print image file.

3.1.6 Batch Mode

The work bench supports a batch mode mechanism to allow for composed operators. This package enables a user to program a sequence of operations in batch programming mode. It definitely provides faster results compared to when a user is attempting the same operations in interactive mode (which is the default menu of the main program). When a user chooses to work in batch mode, menu bar on top also changes bringing in the appropriate menu as applicable for batch mode. The basic reason for speed comes in from the fact that program resorts to no more dialog boxes when operating in batch mode. It is possible to process single image or two images in this mode. Single input image option qualifies a user for selecting portion of the image file for further processing.

A user can build up the *Batch Mode List* by selecting various operations to be performed. All commands listed in the *Batch Mode List* will be executed in a sequential manner. The user can take time between operations to view outputs of the last operation. At any stage during the batch mode session, user can ask for *Batch Default Modifiable Values Dialog Box* to have a look for the input values that have effect on operations currently listed in the batch list. The user can change these values in order to see its effect on the relevant operations. After all operations are over the user can rebuild a fresh batch list for batch mode operations or can load fresh input images and then build the batch list.

3.2 Selection by mouse

This package, enables a user to select a portion of the input image for further processing or saving to disk. For selecting the portion, an user is required to take the mouse pointer to the top left corner of the area to be selected on the input image and click the left mouse button. Thereafter keeping the left mouse button pressed drag the mouse pointer to the other end of the selected portion within the bounds of underlying image. At this point the, hitherto depressed, left mouse button may be released. All along the movement of the mouse pointer from the first point to the last point where the depressed mouse button was released are marked by dotted rectangles as the mouse pointer moves. The dotted rectangle that was formed the last time between the first point and the last point encapsulates the region deemed to have been selected by the user for further processing or saving to disk.

3.3 A Pictorial Tour of the Work bench

In this section a brief glimpse of the various menu items and dialog boxes that are encountered while using this package is given. This package being a Windows application can be run as any other Windows application either from the **File manager** or **Program manager**. Fig 3.1 displays the opening (main) menu as the package is being run.

Before doing any operation it is best to set attributes by selecting the **File | Attributes** menu item. A pull down menu appears as shown at Fig 3.2 for specifying the various attributes. An user can choose the appropriate attribute before doing an operation.

Further, selecting the **Operations | Boolean** menu item brings down pull down menu shown at Fig 3.3. Boolean operations except for Boolean NOT require two input images for processing.

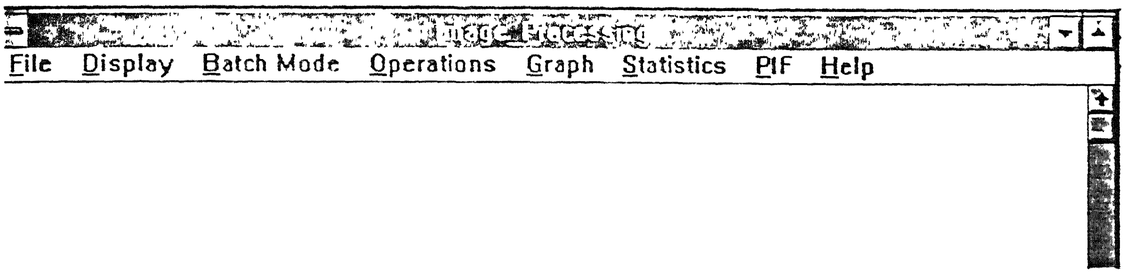


Image Processing
File Display Batch Mode Operations Graph Statistics PIF Help

Fig 3.1: Main Program Menu

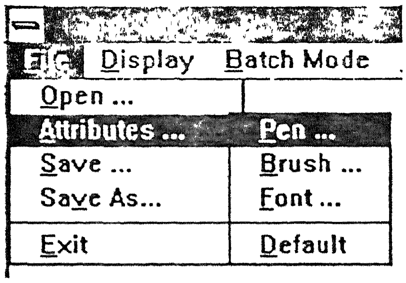


Fig 3.2: Attributes Menu.

Selecting **Operations | Algebraic** menu item brings down a pull down menu as shown at Fig 3.4. Algebraic operations allow both single and double input image operations.

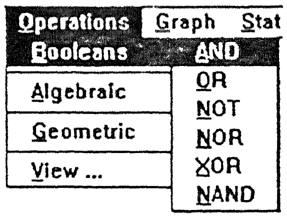


Fig 3.3: Booleans Menuitem.

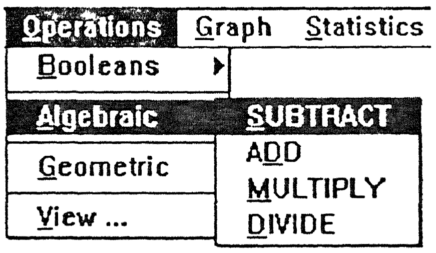


Fig 3.4: Algebraic Menu item.

Fig 3.5 shows the pull down menu that appears once the **Operations | Geometric** menu item is clicked. Selecting **Operations | Geometric |**

Rotation menu item performs the rotation by a specified angle and direction desired (clockwise or anti clockwise), on an input image

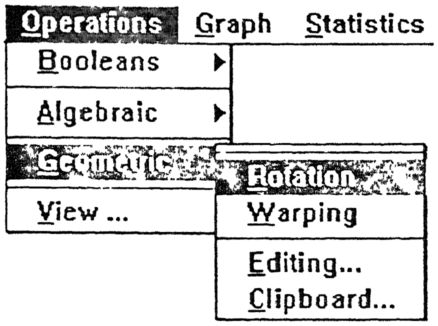


Fig 3.5: Geometric Menu item.

Selecting **Operations | Geometric | Editing** menu item brings up a popup child window which is shown at Fig 3.6 along with its menu. Selecting **Shapes** menu item from the child window brings down a pull down menu as shown at Fig 3.7.

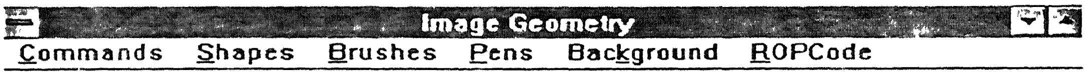


Fig 3.6: Main menu for Editing menu item.

The user can select any shape item from the **Shapes** menu item. These items can be drawn in the window using the mouse. Current settings for Brushes, Pens , Background and ROP code will be effective, which can be varied to see their effects when the shapes are redrawn.

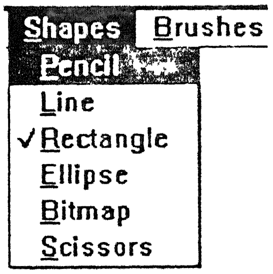


Fig 3.7: Shapes Menu.

Of particular interest are the **Shapes | Bitmap** menu item, which enables a user to load a bitmap image before doing any geometric operations with it.

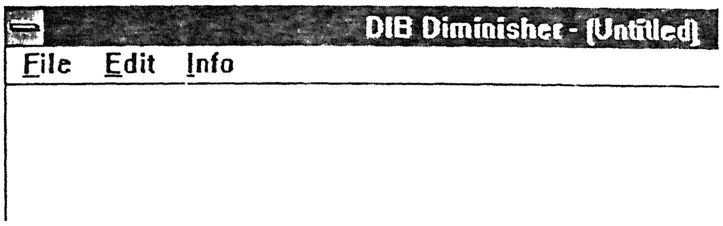


Fig 3.8: Popup menu for Clipboard.

Fig 3.8 shows the menu item that accompanies the popup window as soon as the menu item **Operations | Geometric | Clipboard** is clicked. Clipboard is a Windows way of exchanging data with its applications.

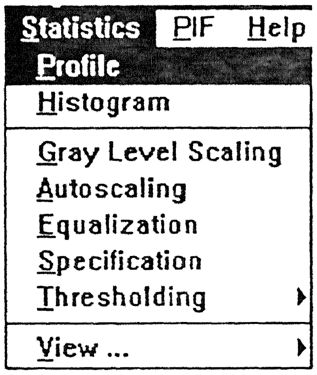


Fig 3.9: Pulldown Menu for Statistics.

Fig 3. 9 shows the pull down menu that appears when the **Statistics** menu item is clicked.

Menu item **Statistics | Profile** enables a user to perform profile analysis. The user can mark the area to be profiled by using the mouse, as explained under *section 3.2, Selection by Mouse*. The user is prompted by the program for *horizontal* or *vertical* direction before going ahead. By clicking menu items **Histogram, Grayscale, Autoscale, Equalization** and **Specification** respective operations are performed.

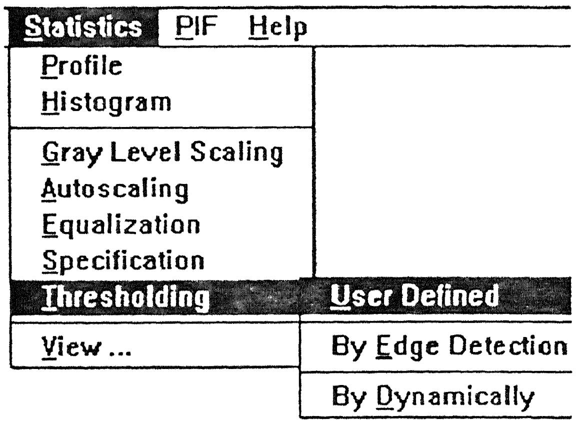


Fig 3.10: Pulldown menu for Thresholding.

Menu item **Statistics | Thresholding | User Defined** enables the user to calculate threshold value and obtain a binary image based on this value. The program computes the histogram of the image and displays it waiting for further user action. The user is expected to locate suitable threshold value from the image histogram, by clicking the right mouse button. The program displays the output in the current window which will be a binary image and its histogram. Menu items **Statistics | Thresholding | By Edge Detection** and **Statistics | Thresholding | Dynamically** are other two ways to threshold an image. Fig 3.10 shows the pull down menu that appears when the menu item **Thresholding** is clicked For more details *User Manual* attached as Appendix D to thesis report may be referred.

Chapter 4

Brief About The Algorithms Used

4.1 Introduction

This chapter briefly discusses the algorithms used in the work bench The algorithms used are standard and can be found in the bibliography at the end. Since the account in this thesis is on building a work bench most of the effort was spent in actually implementing the algorithms in a Windows environment.

4.2 Arithmetic

4.2.1 Addition, Subtraction, Multiplication and Division

```
f(x) Op z(x) -----> g(x)                                        (1)
if g(x) > 255
        g(x):= 255
else if g(x) < 0
        g(x):= 0
```

where f(x) and z(x) are pixel values of two images. **Op** is the operation which can be addition, subtraction, multiplication or division. In case of single image operation, a constant scalar quantity is used as an operand in the above equation. g(x) holds the resultant value of the pixel. In case of division care is taken for the contingency when the denominator is zero. In that case 255 is assigned to the resultant image pixel.

4.2.2 Matrix multiplication

Matrix multiplication is considered to be a specialized operation as it requires that the two images meet the condition for matrix multiplication. Resultant output of the matrix multiplication is not guaranteed to be meaningful as an image to be viewed. This operation is usually performed as an intermediate operation. The algorithm uses unsigned long quantities to deal with the situation, and later scales the result to unsigned char

4.3 Boolean

MS Windows provides a rich set of GDI (Graphical Device Interface) routines for implementing ROP (Raster Operation) Codes. In a painting operation, GDI routines perform a logical combination of three elements: the brush selected into the destination device context (DC), the pixels in the source DC rectangle, and the pixels in the destination DC rectangle.The result is written to the destination DC rectangle. Windows permit a total of 256 ROP codes or combinations. Only 15 most widely used among them have been given formal names. By suitably manipulating the arguments of the GDI routines, it is possible to implement all Boolean operations. If S is the Source bitmap and D is the Destination bitmap The parameter **dwROP** of **DWORD** (Windows way of naming unsigned long) type passed to the API function **BitBlt()** does the needful:

Boolean Operation	**dwROP**	**Remarks**
NOT	NOTSRCCOPY (~S)	Inverts the source bitmap & then copies it to the destination
OR	SRCPAINT (S|D)	Combines the source & destination bitmaps using Boolean OR operator.
NOR	NOTSRCERASE (~(S|D))	Inverts the result of combining the source and destination bitmaps using the Boolean OR operator

AND	SRCAND (S&D)	Combines the source and destination bitmaps using Boolean AND operator.
NAND		Since there is no direct equivalent of NAND operation, it is achieved in a round about fashion. By first ANDing the two images using SRCAND and then finally inverting the destination bitmap using DSTINVERT . DSTIVERT when used as dwROP parameter inverts the destination bitmap (~D).
XOR	SRCINVERT (S^D)	Combines the source and destination bitmaps using Boolean XOR operator

If SRCCOPY is used as dwROP parameter then it simply copies the source bitmap to the destination disregarding the current brush selection.

4.4 Geometric

4.4.1 Rotation

A rotation operation enables a user to rotate an image by a specified rotation angle (assumed to be in degrees) in either clockwise or anticlockwise direction. This algorithm, based on the angle specified and its direction, calculates the new location of the pixel on the bitmap and then copies its color components from its previous location to this location These actions are repeated for each pixel of the image. This algorithm has two cases each working for two kinds of inputs. Case one works if the input angle is 0 or any multiple of 90 degrees. Case two works for any other angle. In the first case only integer multiplication's are involved so they are faster and also gives a true reproduction of the original image. Case two has the advantage that it works for all angles but has a few disadvantages such as: the operation is slower as it involves floating point operations; the output image produced after rotation is not a true replica of the original image.

4.4.2 Scissors tool for Cutting and Pasting

The scissors tool works much like the selection rectangle in Paintbrush (MS Windows application for creating graphical objects) that defines the areas for editing. After choosing Scissors the user drags the mouse to outline an area with a dotted rectangle. Then when the user presses the button again inside the rectangle, the image sticks to the cursor so that it can be dragged to a new location.

Actually the area that the user selects with a dotted rectangle, called the source area, is stored in a global variable of type **RECT** (Windows typedefined struct having two points, begin and end, for a rectangle) called *rSource*. The destination rectangle - the area that sticks to the cursor and moves with it - will be another global variable *rDest*. Given these two variables, the paint procedure moves the image in two steps. First it slides the destination rectangle to a new position, following the mouse, and then it calls **BitBlt** (API function) to copy stationary source image to the current destination coordinates. The source image always comes from the bitmap but the moving copy is another phantom image that appears only on the screen.

After the user selects the Scissors tool, clicking the mouse might mean one of two things. The first click means, "Draw a dotted rectangle", and the next one means, "Move the dotted rectangle". The user drags the mouse once to outline an area, and the next button press initiates another drag operation to move the selected area. The first time button comes up it means something like, "I have finished selecting an area to move", and the second time it comes up it means "Put the source image here". To keep track of the process step by step, to know whether it is time to outline or stick, to select or transfer or copy, as will be seen, a flag variable fScissors is used limited to the following enumerated values:

```
enum {ready, selecting, waiting, copying, cutting, stretching} fScissors = ready;

        ready = Ready to begin cutting and pasting.
        selecting = Marking area to move with dotted rectangle.
        waiting = Rectangle drawn; waiting to begin relocation.
        copying, cutting = fScissors is assigned one of these values depending
                upon the control key if it was down or not, when the user starts
                to drag.
        stretching = when user wants to stretch an image, this value is assigned
                to fScissors if shift key was down before user starts to drag.
```

If the control key is down when the user starts to drag, that means lift a copy and move it. Otherwise, the original image sticks to the cursor. Program uses following macro to determine it.

```
#define     IsctrlKeyDown()     (GetKeyState(VK_CONTROL) & 0x8000)
```

The macro checks the status of the control key by comparing the value **GetKeyState** (which is an API function) returns with the hexdecimal number 0x8000. The result of the logical AND operation is TRUE when the higher bit is on.

When the program first begins to relocate an image, it checks whether the control key is down and set fScissors either to copying or cutting. When the program needs to know whether the user is dragging an image, it will now have to test whether fScissors is greater than copying instead of equal to copying. Following piece of code initializes fScissors for copying or cutting:

```
if (fScissors = = waiting)
        fScissors = IsctrlKeyDown() ? copying : cutting;
```

Each of the different procedures that select, cut, move, and paste updates the scissors flag. Whenever the program needs to know how far the process has already come, it checks fScissors.

4.4.3 Flipping and Zooming

Flip commands can work only if the user has outlined an area with the Scissors. Before the area is outlined, **Flip** menu items are disabled (i.e. grayed). As the area is outlined, *fScissors* is set to *waiting* and **Flip** commands are active.

The task of flipping an image is divided into two procedures. First, *FlipImage* reads the command identifier to determine whether the user chose a vertical or a horizontal flip; calls the second procedure, *FlipBlt*, to invert the image; and finally invalidates the modified area, so the new image will appear.

In order to call *FlipBlt*, *FlipImage* has to set flags defining the direction of inversion. The two flags, *FB_HORZ* and *FB_VERT*, are defined already in the program header file. They are bit values and may be combined with bitwise OR operator to make *FlipBlt* invert the image both ways at once.

If the shift key is down when user clicks on a selected area, program stretches the image instead of moving it. If the button clicks near the right edge, the rectangle stretches to the right. If it clicks near the lower right corner, then the image stretches down and to the right

4.4.4 Preserving Color Palette

Not every adapter supports palettes. The standard Windows 16 color VGA driver does not permit the palette to be changed at all. The VGA is a palette device, but a palette of only 16 colors is quite small. MS Windows requires some colors to remain constant so that, for example, the window frames and caption bars do not change whenever the palette is changed. This leaves only a small space on a 16 color palette for programs to manipulate Under Windows there is a stock palette defined, but instead of 16 colors derived from EGA/VGA RGBI capabilities, Windows defines a stock palette consisting of 20 static colors (the default palette). When Windows applications operate on an EGA or VGA system, only 16 of these 20 colors actually correspond to displayable colors; the remaining four can only be displayed by dithering. Alternatively, on SVGA systems, since the hardware supports a device palette of 256 colors, the 20 default colors appear as individual hues without adjustments.

This program attempts to deal with the situation, by reading the color table from the **BITMAPINFOHEADER** and making a palette out of it. This is achieved in the function *PaletteFromDIB* which returns the handle to the palette created in variable *hPal* (type **HPALETTE**).

GetNumColors routine takes a pointer to **BITMAPINFOHEADER** and returns the number of colors (i.e. color table size) in the current DIB. **GlobalAllocPtr** is an API macro which allocates memory for **LOGPALETTE** (logical palette) structure. After initialization, API routine **CreatePalette** creates the palette and returns handle to it in *hPal*. **GlobalFreePtr** is another API macro that frees the memory when it is no longer needed. *hPalDIB* is a global variable of type **HPALETTE** which has the handle to the logical palette and is accessed by routines that process messages send by the Windows when a logical palette is realized.

Two other routines need a mention here as they directly deal with Windows messages pertaining to change in palette. First routine is *Dibdim_OnQueryPalette* which processes the message

WM_QUERYNEWPALETTE message by realizing our palette (DIB's palette) as our program receives the input focus. **WM_QUERYNEWPALETTE** message arrives when the window is becoming active and offers the opportunity to choose the colors we want by realizing a palette. Other applications may have put different colors into the system from their images and we may need to reset them. Second is *Dibdim_OnPaletteChanged* routine which processes the other Window's message **WM_PALETTECHANGED** by realizing our palette and calling API routine **UpdateColors** to make quick, reasonable matches with the new system palette. When a **WM_PALETTECHANGED** message arrives, the program selects the palette into the DC and tells Windows to realize those colors, meaning to find the best matches it can on the current device and put them into the system palette. Any subsequent GDI function calls will use the new colors. **UpdateColors** work by considering what color each pixel mapped before the palette changed and finding the best approximation in the new palette. Because each new update is based on the previous approximation, the picture deteriorates with successive updates. *Dibdim_OnPaletteChanged* avoids excessive deterioration by counting the updates and forcing a periodic repaint. Internally each palette object remembers three distinct ways to map its colors into the system palette: how they are currently mapped, how they were previously mapped, and how they will always be mapped whenever the palette has foreground priority.

Despite all this no palette manipulations will have any visible effect with the standard 16 color VGA driver. The primary limitation is physical. If the device supports only 16 colors (as for 16 color VGA driver), then only 16 custom colors can be displayed and all the remaining palette entries will be mapped to the 16 supported colors. In like fashion, for an SVGA system, a physical palette limitation of 256 colors is imposed by the system hardware.

4.4.5 Run Length Encoding

The routine *ReformatDIB* is called whenever the user clicks on one of the RGB/RLE radio buttons in the Measurements Dialog Box. *ReformatDIB*, with the help of conversion procedures that follow, makes a copy of the DIB and changes the compression setting. If the first bitmap is compressed, the copy will be expanded; if it is already expanded, the copy will be compressed. API routine **GetDIBits** can render an image in compressed or uncompressed format. **GetDIBits** converts a DDB to its corresponding DIB, and if one asks for a compressed result it is obtained. The primary requirement is conversion of a DIB to DDB. Which means it has to be translated into a DDB before we can call **GetDIBits** to convert it back into a reformatted DIB. The major steps are:

- Convert the current DIB into a DDB. It is achieved through a procedure *DIBtoBitmap*. Every device dependent bitmap (DDB) needs a device to be dependent on, so the conversion procedure needs to be given a DC.
- Fill in a **BITMAPINFOHEADER**. In order to convert a DDB to a DIB, specification has to be made of the kind of DIB desired keeping in mind several kind of DIB's with different color capabilities. The values in the **BITMAPINFOHEADER** structure describes the bitmap to be created.
- Convert the DDB back into new DIB with a new format. It is achieved through the routine *BitmaptoDIB*.

There is an inevitable problem here, if the DIB uses for example 256 colors, and if it is translated into a device that shows only 16 colors (as in case of 16 color VGA) or perhaps only two then the extra colors are lost. *Reformat DIB* procedure after getting a handle to the DDB bitmap obtained from from *DIBtoBitmap* makes relevant changes in the **BITMAPINFOHEADER**. If the original DIB held 256 colors or graylevels but screen uses only 16, in that case the **biClrUsed** field is initialized to 16. Also the **biBitCount** is initialized to 4 and not 8, if the video hardware supports only 16 colors. Besides, **biCompression** field must be toggled to another value. Variable *bih* (of type **BITMAPINFOHEADER**) after being filled with above information is passed to the conversion procedure *BitmapToDIB* with other parameters to finally convert the image to a reformatted DIB.

Of particular interest here is the API routine **GetDIBits**. It is called twice. First time it is called with a NULL parameter for the image bits. **GetDIBits** returns the projected size and put colors from the system palette into the color table of the new DIB, so if the old DIB's palette is realized before converting (as is the case here), we get the same colors back (or atleast the best matches the intermediate screen driver supports). Second time **GetDIBits** is called with all all it parameters and it copies DDB into a DIB.

4.4.6 Clipboard

The ability to share data between programs make computers more productive, and MS Windows supplies several avenues of data exchange. One of them is the clipboard, which lets the user cut information from one program and paste it into another without closing either one. This package enables a user to use **Cut** and **Copy** commands to give information from the program to the clipboard. **Cut** actually deletes the information from the program; **Copy** leaves the original intact. The **Paste** command moves information from the clipboard into the program.

Windows come with a utility program called **clipbrd.exe** that shows in its own window what the clipboard contains. The system clipboard and Clipboard Viewer utility programs are two different entities. It is possible to Cut and Paste without ever running **clipbrd.exe**. This utility finds its mention for the users of this package because it performs a useful diagnostic function for testing a programs clipboard capabilities. As this implements clipboard logic, this diagnostic utility will be able to test its clipboard capabilities. Its Display (Clipboard Viewer Utility) menu item lists all the data formats currently available. For example Fig 4.1 shows the data formats this package offers.

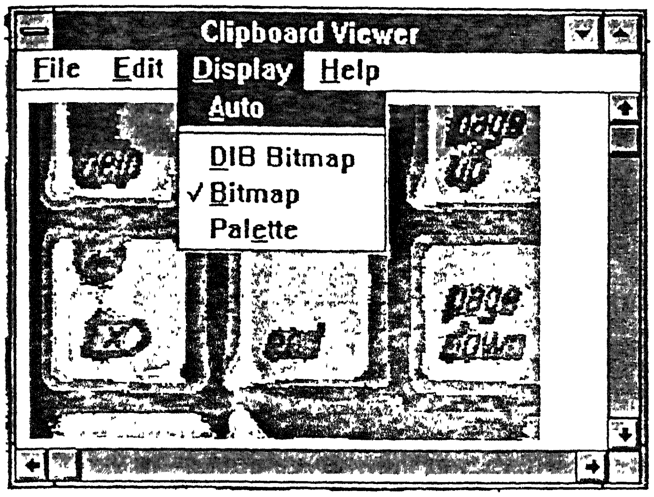


Fig 4.1: Clipboard Viewer utility showing the data formats supported by the program.

Although there are numerous formats for data exchange through clipboard. In particular for the purpose of this package, which is an image processing application, DIB and DDB formats are implemented in addition to support of palette from its current data.

A program can make any of three transactions with the clipboard: it inquires about the available formats, take data from the clipboard, or give data to it. All three operations begin with the same command: **OpenClipboard** (an API routine). Only one application may open the clipboard at a time. The clipboard remains open until it is explicitly closed by giving a call to another API routine **CloseClipboard**, which makes the clipboard facility available to other programs. Information copied to the clipboard remains there until new data replaces it. After opening the clipboard, any program that wants to put new data there should first call **EmptyClipboard** (an API routine) to clear old objects. **EmptyClipboard** destroys the current contents, freeing memory. The user, not the program, controls the flow of data through the clipboard. Information is destroyed only when the user decides to replace it with something else. In order for this to work the clipboard, not the program, must own the data it receives.

There is a way to tell the clipboard what the program has without actually giving it any data. It can be achieved by setting NULL value in the API routine **SetClipboardData** for each available format, as follows.

The NULL values promise the clipboard that the program will provide DIB and DDB data objects. When the user tries to paste these objects into another program, Windows sends a message called **WM_RENDERFORMAT**. Only after an object has actually been requested that the package makes a duplicate copy of the objects requested for the clipboard. This technique is called delayed rendering. It avoids making an unnecessary copy of a bitmap before it is actually required. A program that delays rendering clipboard objects must always preserve a copy of anything it has promised to the clipboard. Even when the user loads bitmap B, the earlier bitmap A (say) that was promised to the clipboard should be held on. Windows sends another message, **WM_DESTROYCLIPBOARD**, when the user changes the clipboard contents. Infact, **EmptyClipboard** will send this message to the previous donor (of the clipboard). On receipt of that message the program can destroy the old copy of bitmap A. The user has placed some new object on the clipboard, and the program can no longer be asked upon for the old one. The program declares a global variable *lpvClipDIB* (of type **LPVOID**) for remembering what the user put on the clipboard. After a successful **Cut** or **Copy** operation, program remembers to update the global variable and a Boolean flag.

Another action, that an application which resorts to delayed rendering, should take is to process another Windows message **WM_RENDERFORMATS**. It is possible that the user closes the program that has promised data to the clipboard. Under these circumstances, this message is received by the application. In response to that message program should convert the data promised to every available format and give the clipboard every object the clipboard can .

There are two routines *CopyBits* and *PasteBits*, besides the message handlers, that deals with copying and cutting to the clipboard and pasting from the clipboard respectively.

As mentioned delayed rendering means coping with three messages. On **WM_RENDERFORMAT** the program provides the object in the format requested. On **WM_DESTROYCLIPBOARD** the program releases any old data preserved for the clipboard. On **WM_RENDERALLFORMATS** program provide copies of the data in every available format. A new procedure handles each message. When the user wants to paste our bitmap then our program must provide a copy in the correct format, either **CF_DIB** or **CF_BITMAP**, and put the handle on the clipboard.

- Taking a bitmap from the clipboard involves these steps:
- Open the clipboard.
- Take the bitmap handle from it
- Make a copy of the clipboard bitmap.
- Close the clipboard.

PasteBits routine first ascertains the object at the clipboard, it may be either DIB or DDB, and then calls *PasteDIB* or *PasteDDB* to create a private copy of the clipboards bitmap. In either case, if *PasteBits* successfully acquires a pointer to the new bitmap it destroys the old bitmap with *ClearDIB*. Only when the new bitmap is secure and the old one destroyed does the global *lpvDIB* get a new value. Finally the procedure displays the new image.

Procedure *PasteDIB* is called when the clipboard contains a DIB. Procedure *DuplicateDIB* does the actual copying. *PasteDIB* locks the clipboard object, verifies that it is not a OS/2 bitmap, and make a duplicate. If the clipboard contains a DDB than it has to be converted to a DIB format while it is copied. Procedure *BitmapToDIB* does that. In order for it to work, though, *BitmapToDIB* needs to be fed a **BITMAPINFOHEADER** describing the DIB we want to create. API routine **GetObject** helps in filling fields of the **BITMAPINFOHEADER**. Then it checks whether the clipboard has a palette to go with the bitmap. Finally it calls, *BitmapToDIB* and returns the resulting pointer.

4.4.7 Printing the DIB

This package implements printing through banding as it improves system performance for programs that print bitmaps. It makes the most difference when the output involves large block transfers to a device with no high level language for representing graphics. This includes most dot matrix printers. Banding not only saves memory by working with pieces instead of the whole page, it also saves disk space

Procedure *PrintMain* performs the standard house keeping. It calls procedure *GetPrinterDC* to obtain DC for the printer. Then sets the abort procedure *AbortTestProc*, followed by creating a modeless Cancel dialog box which appears at the center of the screen through procedure CentrePop. API routine **EnableWindow** eliminates user input. It is used as a safeguard to prevent user from asking to print another document in the middle of the other one, deleting the object that is being printed or quitting the program. This can happen especially when the abort procedure has been called which allows other applications to run including this one while printing is underway. Procedure *PrintDIB* is called to print the current bitmap A second call to **EnableWindow** restores the original status followed by the destruction of the Cancel dialogbox and printer DC when the printing is over. **FreeAnyProcInstance** is an API macro that enables releasing of the instance bindings of the Cancel and abort procedures from the current instance of the program.

PrintDIB procedure does the actual printing. It sends the bitmap to the printer in a banding loop. Here are the main steps:

- Calculate the scaling factors, source and target boundaries.
- Initialize the **DOCINFO** structure with a string naming the print job.
- Call API function **StartDoc** to begin printing.
- Call **NEXTBAND** over and over, sending each band to the printer with the API function **StretchDIBits**. Stop if the user has clicked over the Cancel button, if **NEXTBAND** produces an error, or after the last band is printed.

- Call the abort procedure each time through the loop to yield processor time for other programs
- Call API function **EndDoc** to finish printing, or an error handler, if anything went wrong

4.5 Statistical, Grayscaling and Thresholding routines

These are routines algorithms for which have been taken from existing reference material. They have been appropriately adapted to be used with the work bench Thus, these algorithms, are briefly discussed here.

4.5.1 Profiling

This algorithm sums up the pixel values, of a selected region, along a specified direction (horizontal or vertical) and then gives a corresponding plot of the selected region, after suitable scaling, and superimposed on the selected region.

4.5.2 Histogram

The histogram of an image is obtained by finding the number of pixels at a particular gray level for all the graylevels in the image. Number of pixels obtained at each graylevel is further divide by the total number of pixels in the image to get its histogram.

4.5.3 Linear and Non linear Grayscaling

Graylevel scaling remaps the graylevels within an image by applying a transformation function to it. In linear transformation functions brightness and contrast parameter applies uniformly to the whole image. In the non linear case it is possible to apply a contrast factor to a restricted portion of the original image. Linear transformation function $\mathbf{g = c * f + b}$ is used. $\mathbf{f}$ is the value of the original pixel, $\mathbf{g}$ is the transformed value of the same pixel. Variable $\mathbf{c}$ varies the contrast and variable $\mathbf{b}$ varies the brightness in the output image. For the non linear case the transformation function used is $\mathbf{g = 31.875 * \log2(f + 1)}$, where $\mathbf{f}$ and $\mathbf{g}$ have the same meanings as for the linear case. This particular non linear transformation function increases the contrast of dark regions and at the same time decreases the contrast of brighter portions within an image.

4.5.4 Autoscaling

The autoscaling operation scales the graylevels within an image to the dynamic range of the system being used. It uses the linear transformation mapping function $g = \{255/(\mathbf{fmax} - \mathbf{fmin})\} * (\mathbf{f} - \mathbf{fmin})$ for a 256 graylevel imaging system. Where **fmax** and **fmin** are the maximum and minimum graylevels respectively in the image and **f** is a particular graylevel in it. The perceived effect on the image is to increase the overall brightness and contrast in the image.

4.5.5 Equalization

Histogram equalization uniformly redistributes the graylevel values of pixels within an image so that number of pixels at any one gray level is about the same. The transformation function is $\mathbf{g_i = (m - 1) \sum^{i}_{j=0} h_{fi}}$ Where g_i is the graylevel of the pixel in the equalized image, $\mathbf{h_{fi}}$ is the histogram of the original image, **m** is the number of graylevels in the original image and **i** is the **i**th graylevel for the equalized image. The histogram for a histogram equalized image is a uniform histogram.

4.5.6 Histogram Specification

Histogram Specification allows an user to specify the final histogram. Histogram Specification can be used to darken an image or, to brighten and improve the contrast of an image interactively by specifying the desired histogram.

4.5.7 Thresholding by User Defined threshold

The program displays the histogram of the input image. The user can suitably choose a thresholding point, preferably between any two prominent peaks on the image histogram. This threshold value is used globally to decide the pixel value of every pixel in the image. The pixels having values greater than it are assigned a graylevel of 255 and those having a value lower than this threshold are assigned a graylevel value 0.

4.5.8 Thresholding by Edge Detection

The algorithm performs edge detection on the image using the Sobel operator. Then it sums up all the pixels on the detected edges and computes its average graylevel. This average graylevel value is taken as the threshold value and applied globally to the entire image to obtain a binary image.

4.5.9 Thresholding using Dynamic operator

This technique is a local technique and it finds the threshold value for every pixel in the image and based on that threshold value decides the final value of that pixel in the thresholded image. The threshold level for a pixel is based on a 3 * 3 neighborhood. The formula is:

byThres = 0.18 * dStdDev * dMean;

Where **dStdDev** is the *standard deviation* of the neighborhood area and **dMean** is the *mean*. The threshold value, *byThres*, found is compared with the value of the pixel to give it a value of 0 or 255 in the binary image.

Chapter 5

Results

This chapter presents the results of the operations performed by the work bench. The results are in the form of images which result when the operation mentioned is performed. The raw images used in the testing are shown in the next section.

5.1 Images used

The raw images used in this chapter are shown in figures 5.1 to 5.5. These images in the .BMP format were the input images to the work bench.

- *panic.bmp*, 256 graylevels, size 262 * 245.
- *sultry.bmp*, 16 graylevels, size 297 * 256.
- *bridge.bmp*, 256 graylevels, size 320 * 200.
- *mat1.bmp*, 16 graylevels, size 80 * 80.
- *mat2.bmp*, 16 graylevels, size 80 * 80.

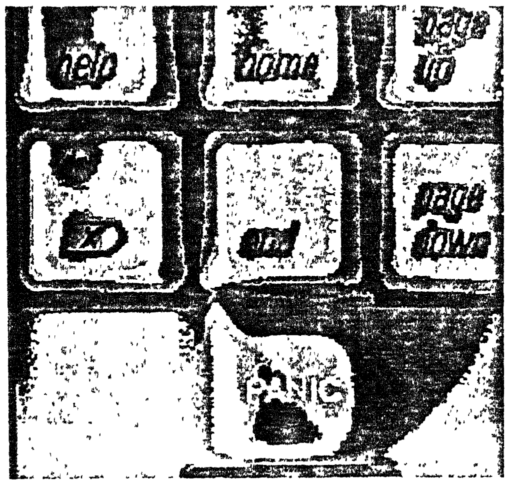


Fig 5.1: Image panic.bmp.

Fig 5.2: *Image sultry.bmp.*

Fig 5.4: Image mat1.bmp.

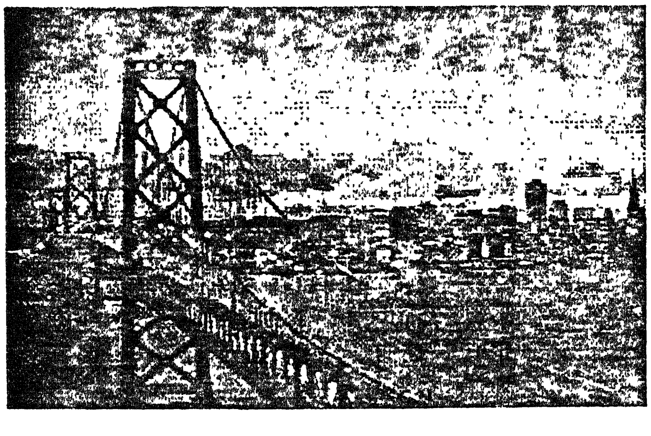


Fig 5.3: Image bridge.bmp.

Fig 5.5: Image mat2.bmp.

5.2 Arithmetic Operations

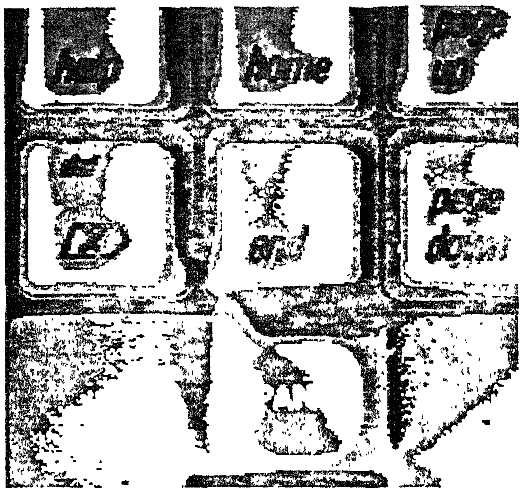


Fig 5.6: Adding value 50 to panic.bmp.

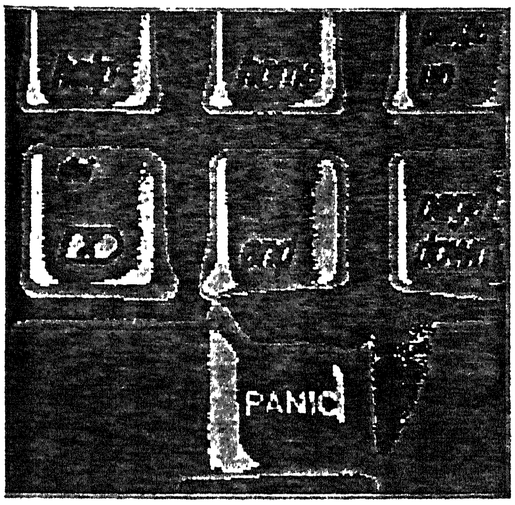


Fig 5.7: Subtracting value 50 from panic.bmp.

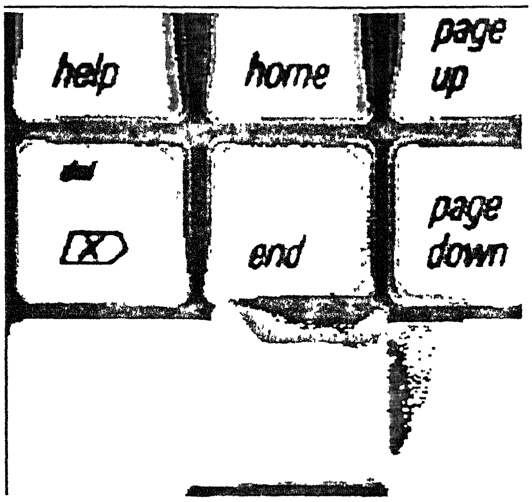


Fig 5.8: Multiplying panic.bmp by 2.

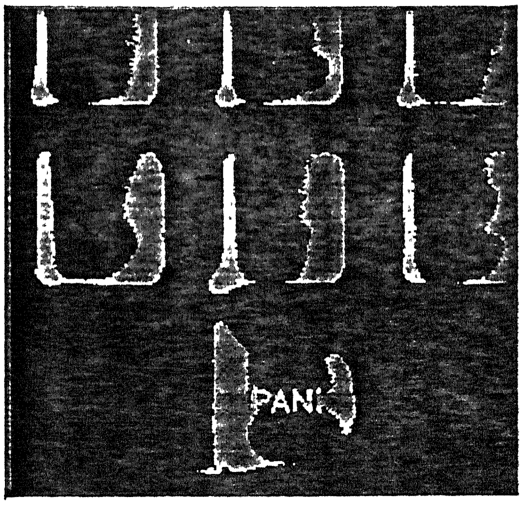


Fig 5.9: Dividing panic.bmp by 3.

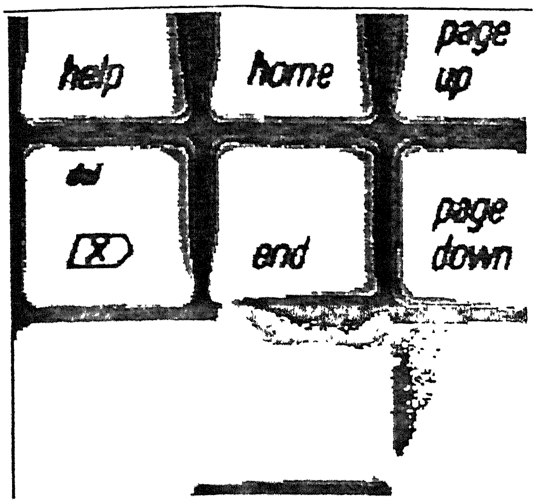


Fig 5.10: Adding panic.bmp to itself.

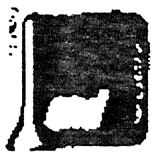


Fig 5.11: Subtracting mat1.bmp from mat2.bmp.

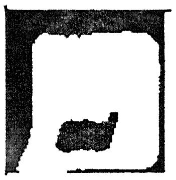


Fig 5.12: Multiplyi mat1.bmp to mat2.bmp.

5.3 Boolean Operations

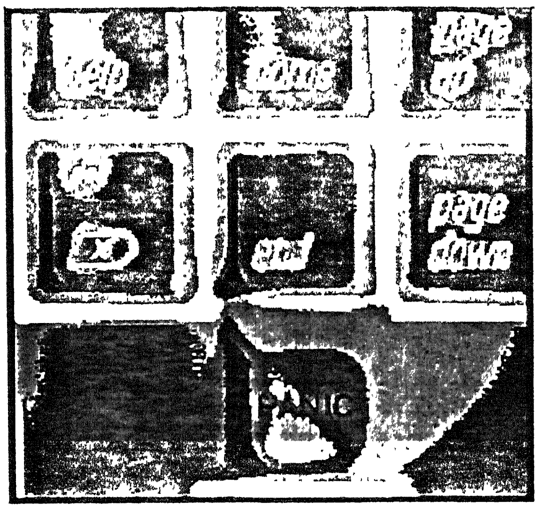


Fig 5.13: Boolean NOT of panic.bmp.

Boolean NAND and NOR operations on two copies of image *panic.bmp* yields similar to Boolean NOT output shown in the last figure. Boolean XOR operation on two copies of image *panic.bmp*, yields a uniformly dark image.

5.4 Geometric Operations

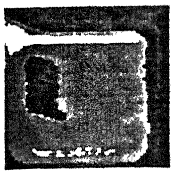


Fig 5.14: Rotated mat1.bmp by 90 degrees

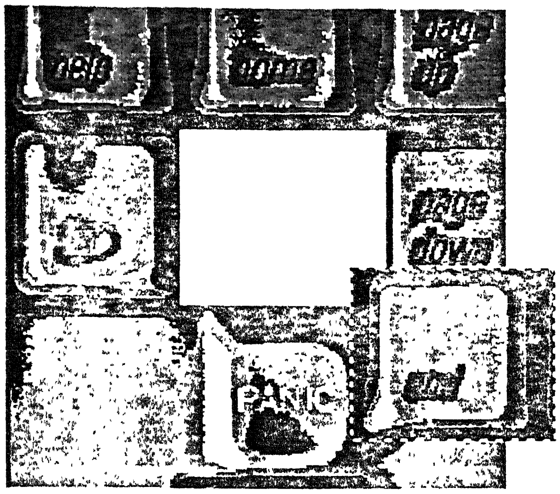


Fig 5.15: Cutting and dragging by mouse.

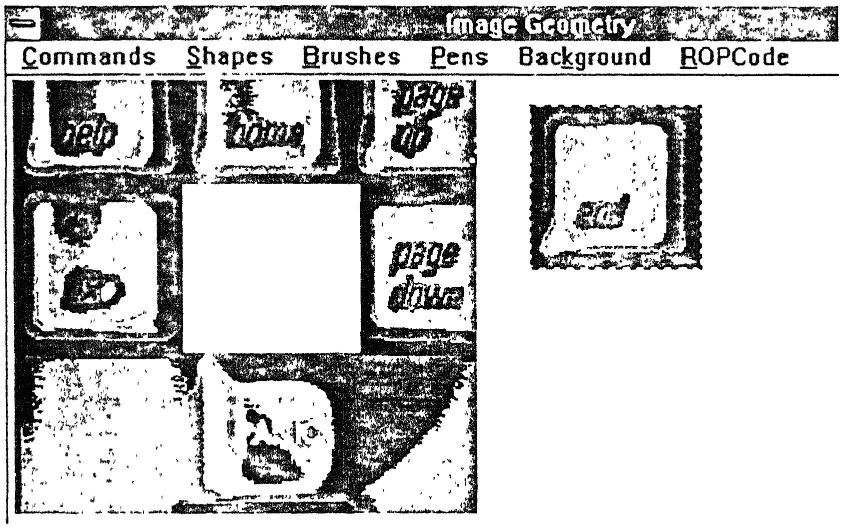


Fig 5.16: Cutting and Pasting.

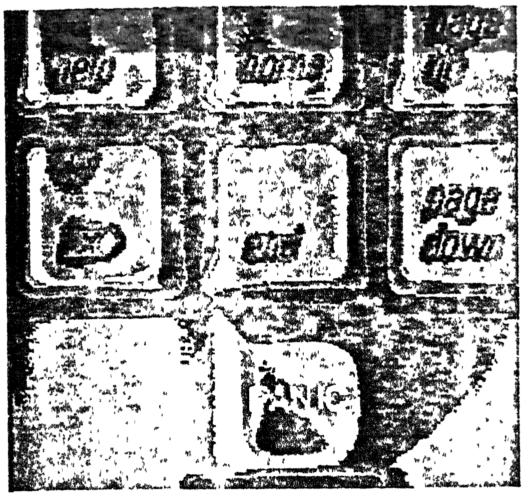


Fig 5.17: Copy & Paste.

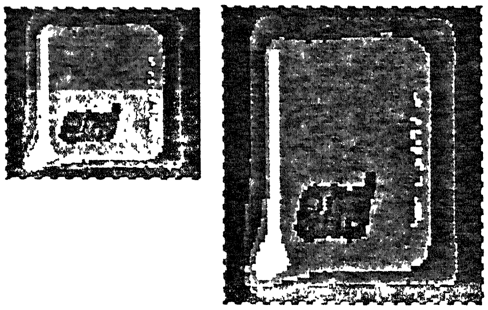


Fig 5.18: Zooming selected portion.

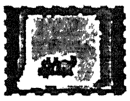


Fig 5.19: Dezooming

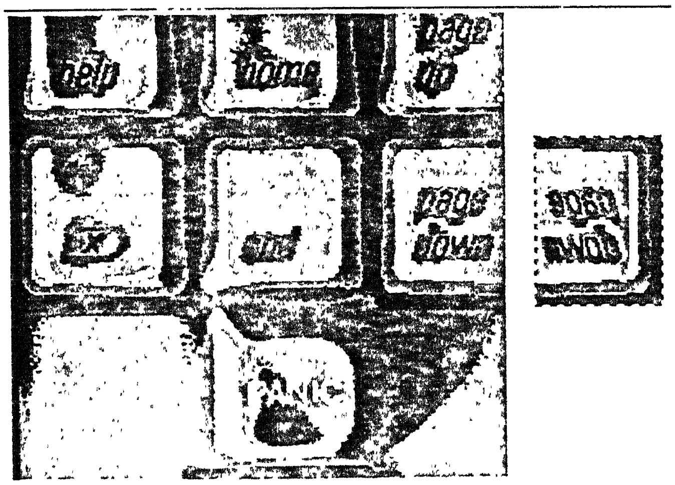


Fig 5.20: Flipping Horizontaly.

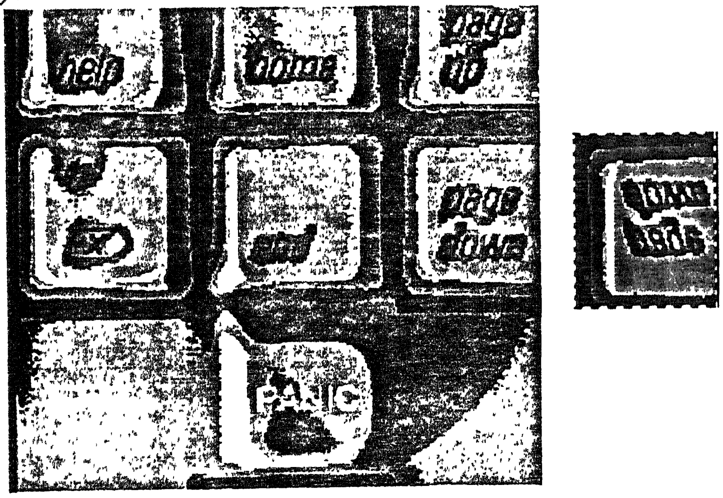


Fig 5.21: Flipping Vertically.

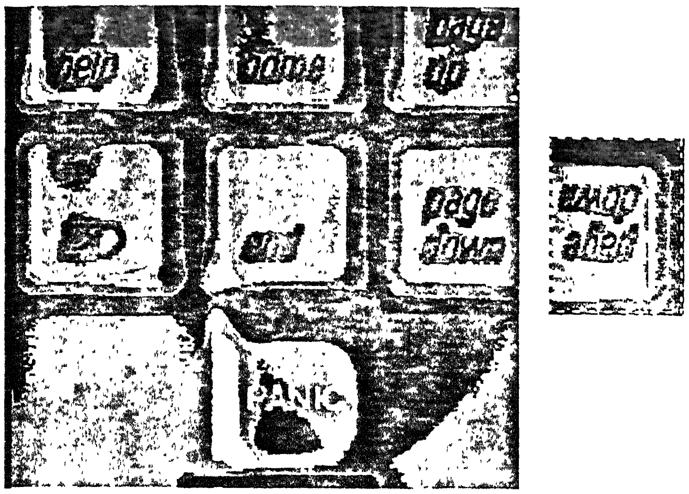
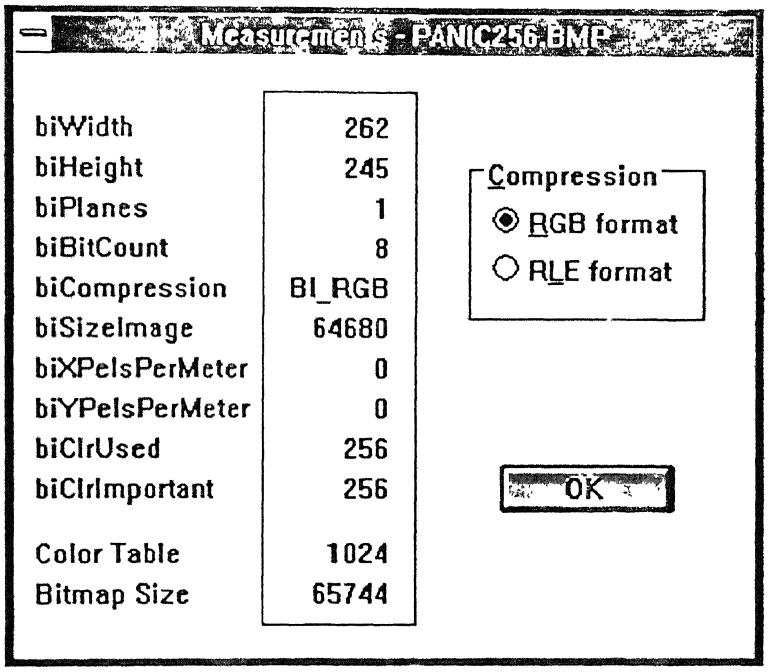


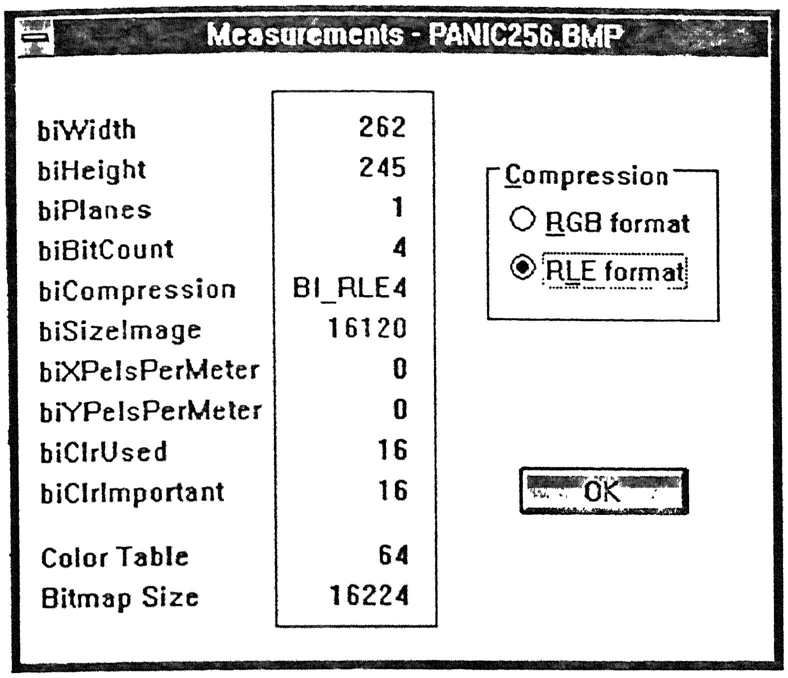
Fig 5.22: Flipping both ways.



biWidth	262
biHeight	245
biPlanes	1
biBitCount	8
biCompression	BI_RGB
biSizeImage	64680
biXPelsPerMeter	0
biYPelsPerMeter	0
biClrUsed	256
biClrImportant	256
Color Table	1024
Bitmap Size	65744

Compression
[✓ RGB format]
RLE format

Fig 5.23: Specification dialogbox for panic.bmp.



Measurements - PANIC256.BMP

biWidth	262
biHeight	245
biPlanes	1
biBitCount	4
biCompression	BI_RLE4
biSizeImage	16120
biXPelsPerMeter	0
biYPelsPerMeter	0
biClrUsed	16
biClrImportant	16
Color Table	64
Bitmap Size	16224

Compression: RGB format [✓ RLE format]

Fig 5.24: Dialogbox after the panic.bmp is compressed.

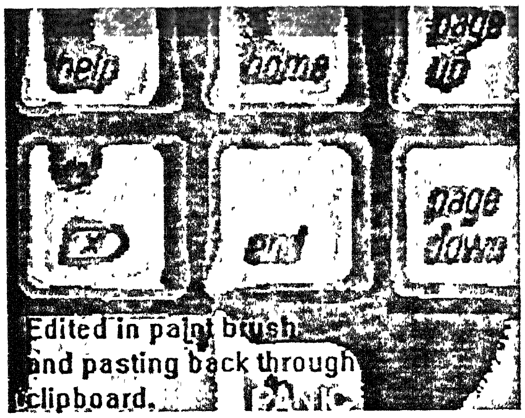


Fig 5.26: Image panic.bmp copied to paintbrush using program clipboard facility; edited and copied back into the program.

In Windows paintbrush application, images can be edited. Recopied to Clipboard and again brought to the program, which can be compressed in the program for saving on disk. This feature is not available to those programs which do not implement clipboard facility as part of their logic.

5.5 Statistical Operations

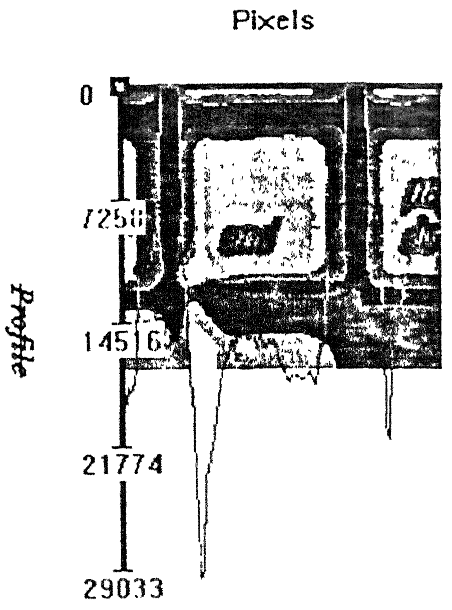


Fig 5.27: Selected portion of panic.bmp, profiled, horizontally.

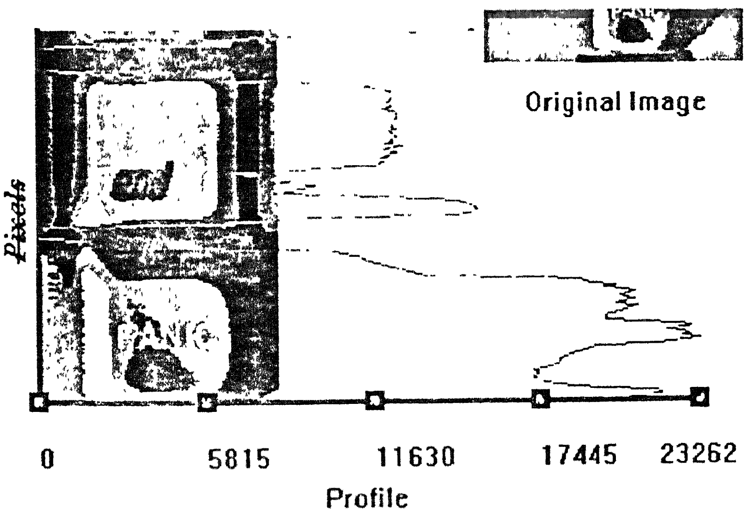


Fig 5.28: Selected portion of panic.bmp, profiled, vertically.

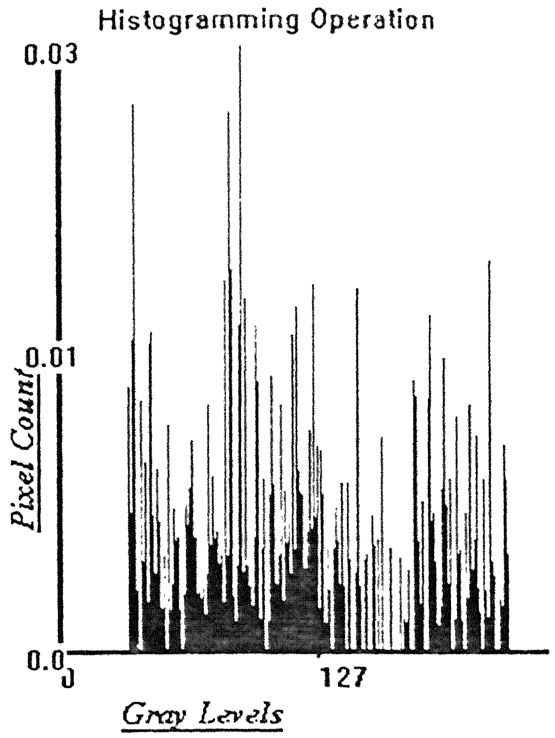


Fig 5.29: Histogram of bridge.bmp.

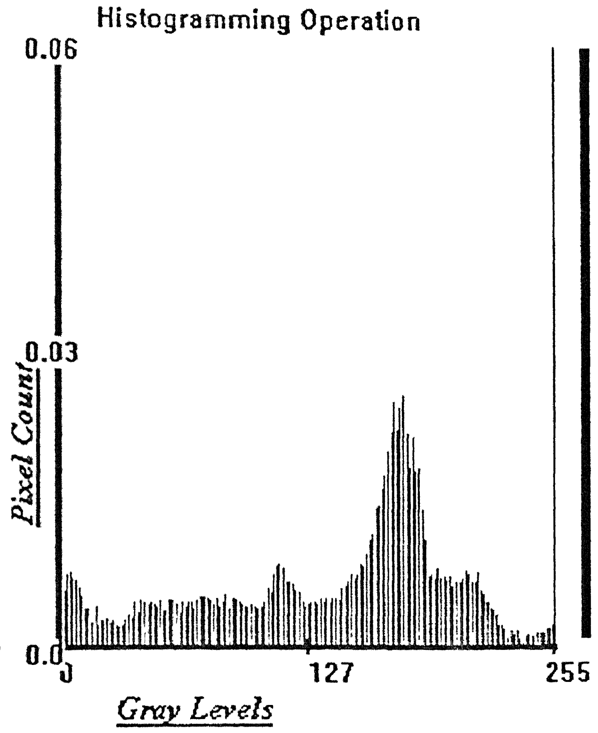
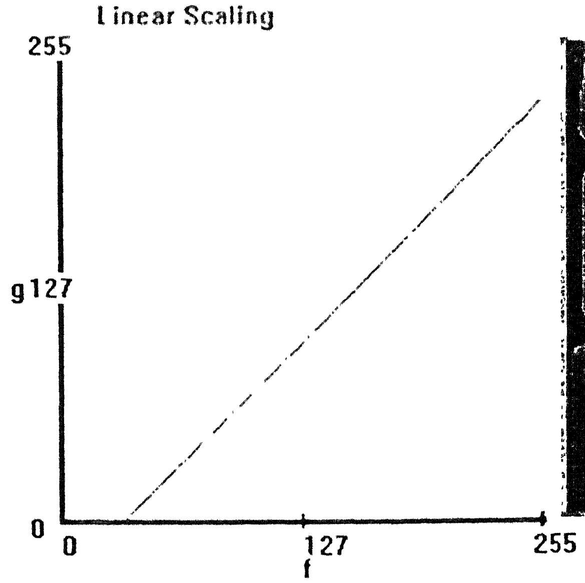


Fig 5.30: Histogram of panic.bmp.

5.6 Grayscaling Operations



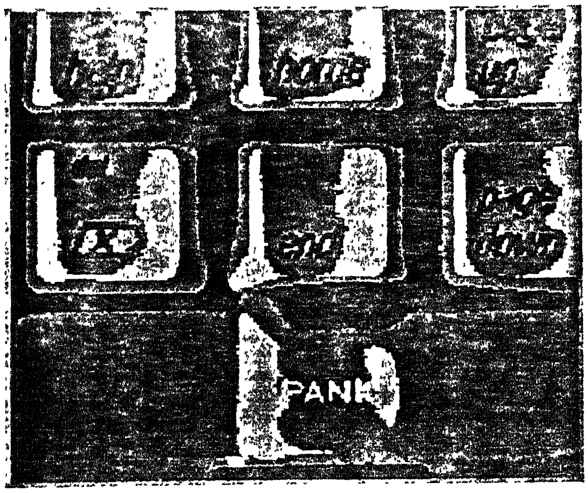


Fig 5.31 : Linear Grayscaling of panic.bmp, for b = -32 and c = 1.

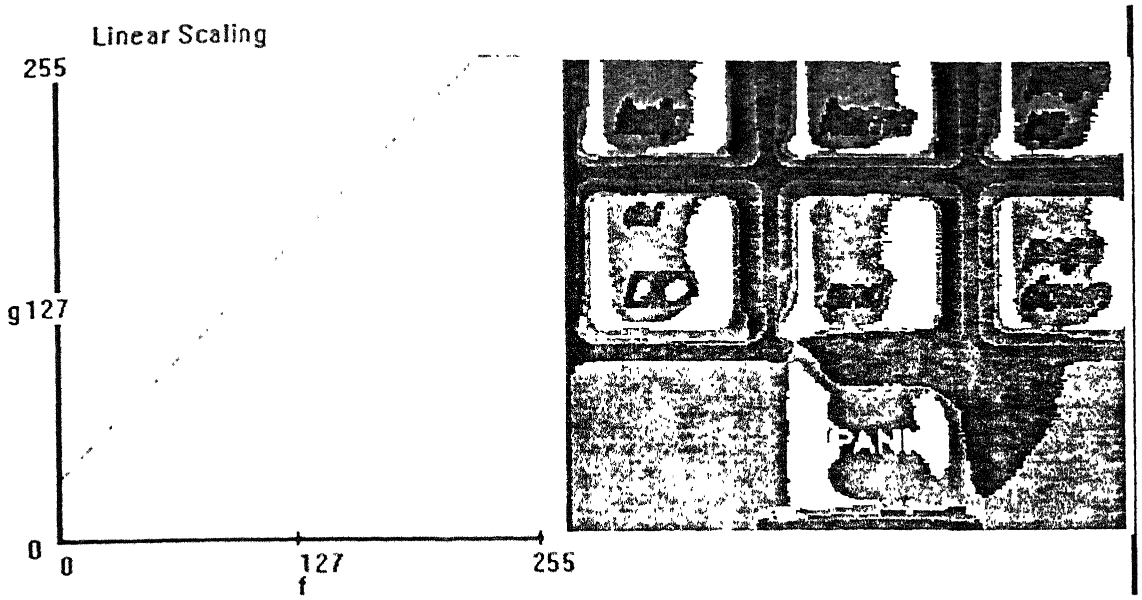


Fig 5.32 : Linear Grayscaling of panic.bmp, for b = 32 and c = 1.

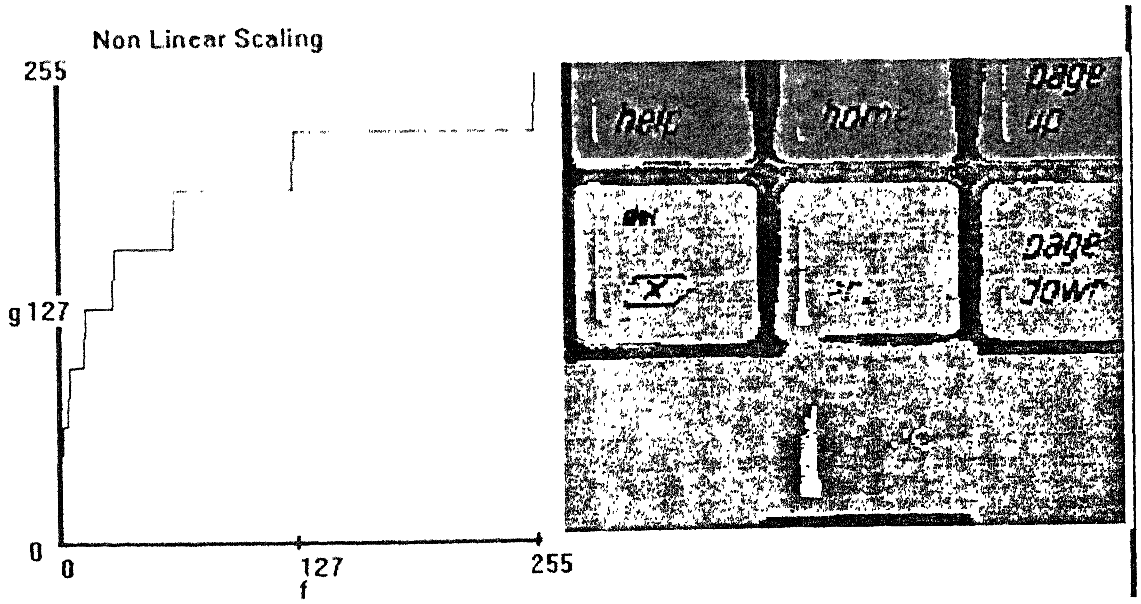


Fig 5.33: Nonlinear grayscaling on panic.bmp, for c = 31.875 and b = 1.

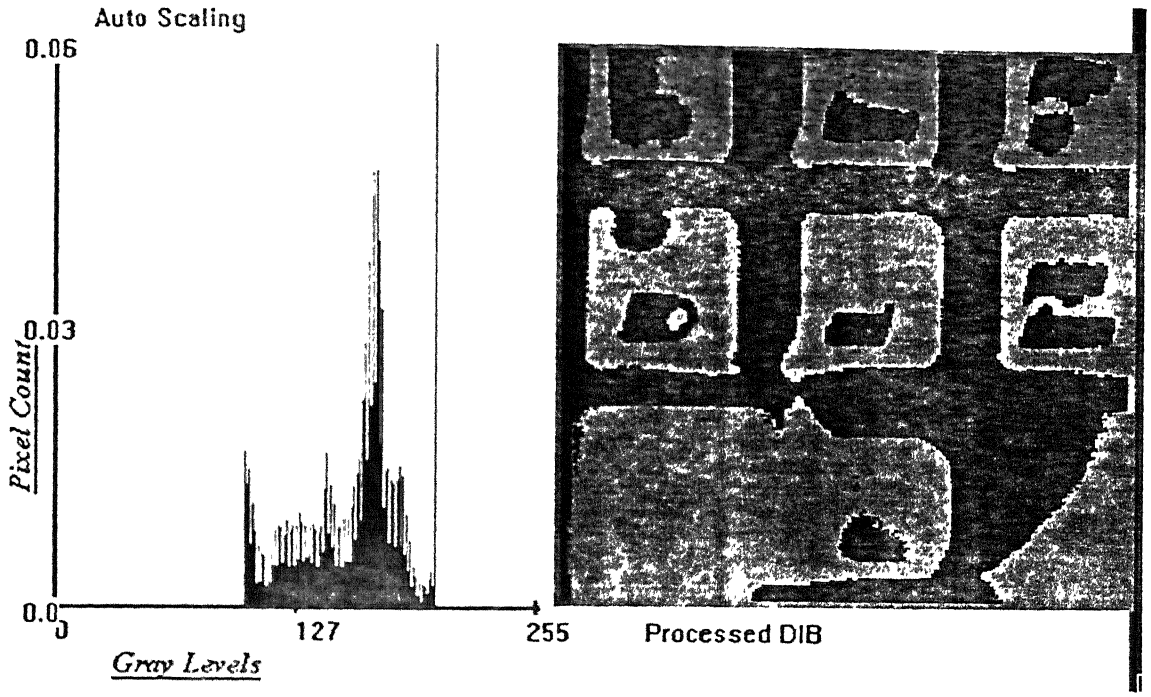


Fig 3.34: Autoscaling on panic.bmp, for graymin = 100 and graymax = 200.

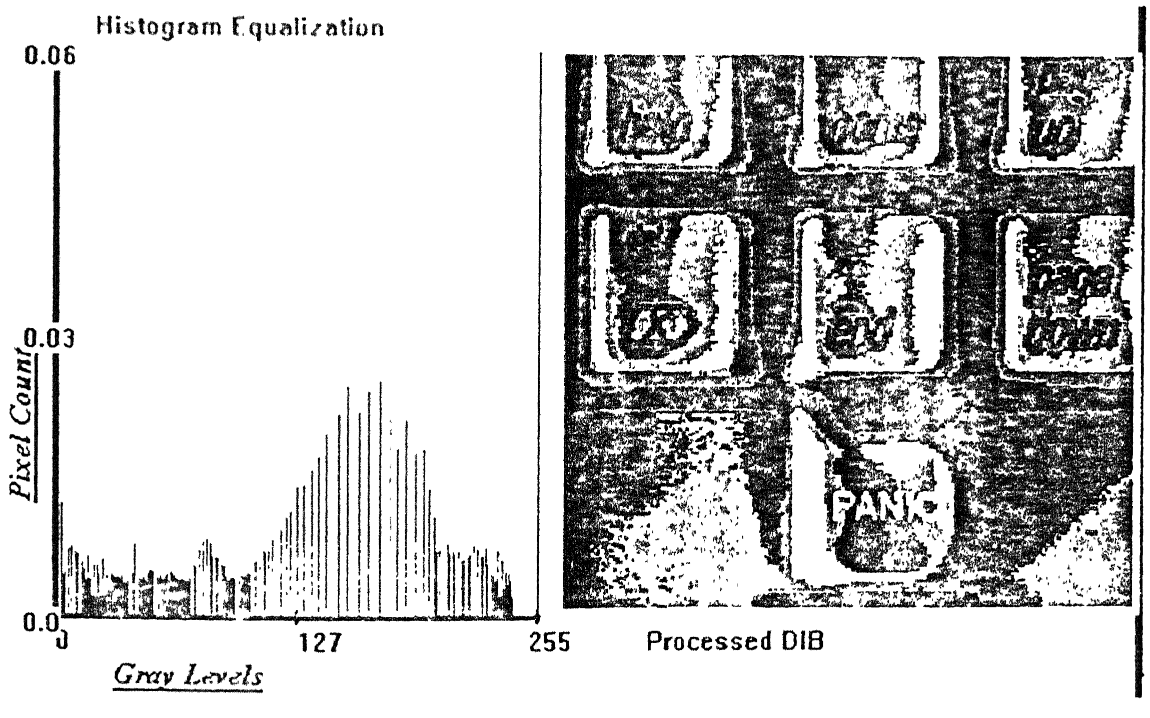


Fig 5.35: Equalization of panic.bmp.

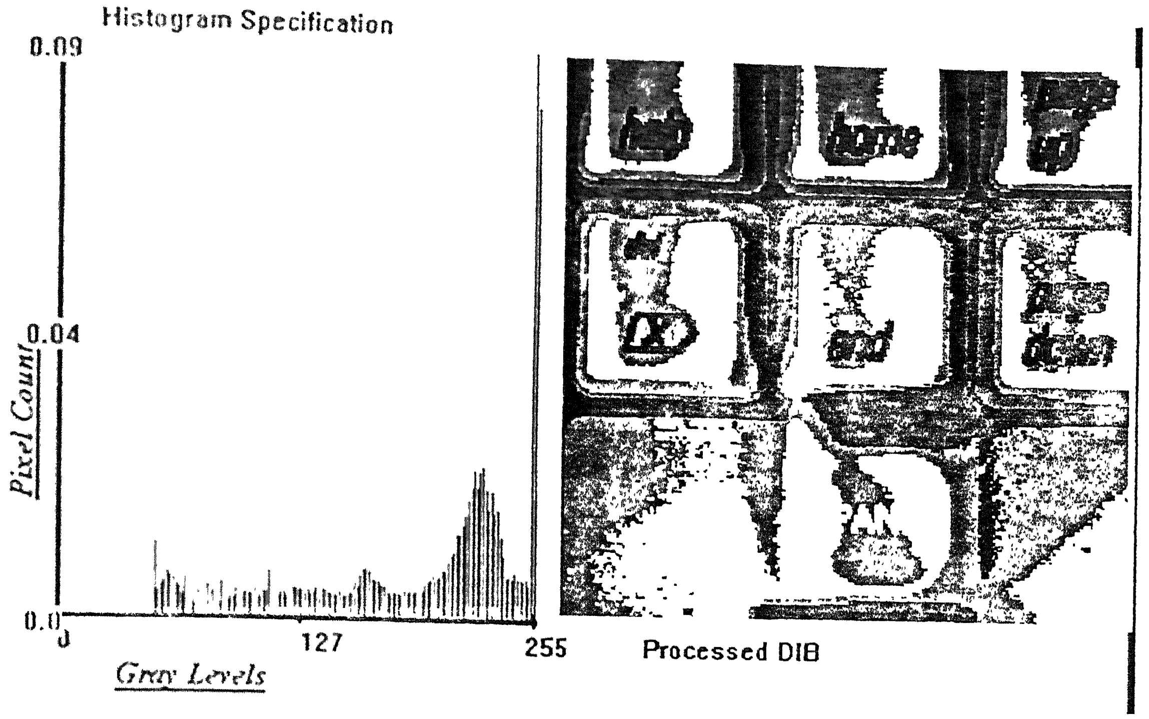


Fig 3.36: Histogram specification on panic.bmp, for a brighter histogram.

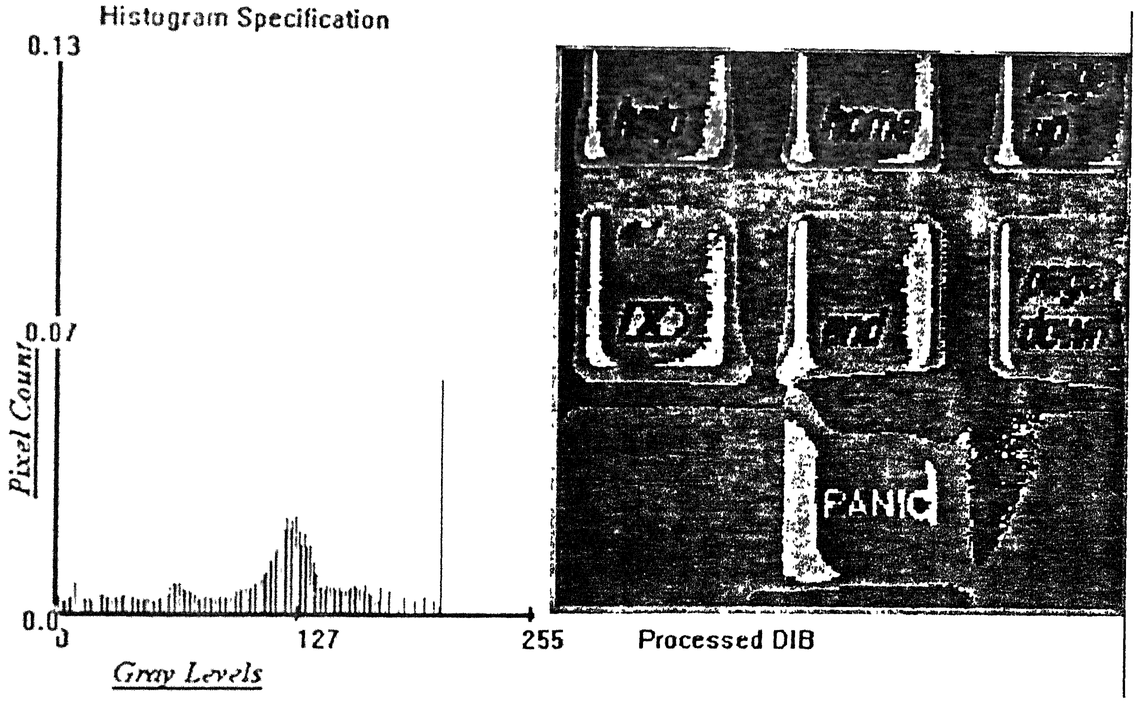


Fig 5 37: *Histogram specification on panic.bmp, for a darker histogram.*

5.7 Thresholding Operations

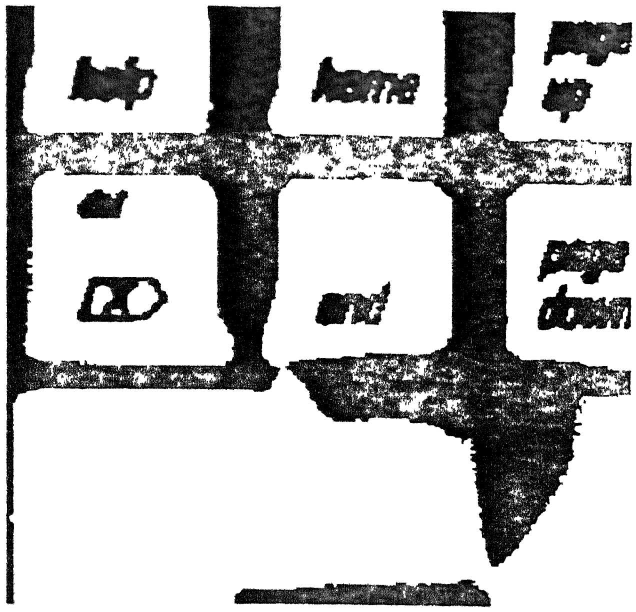


Fig 5.38: Binary image of panic.bmp, using user defined case, for a threshold value of 126.

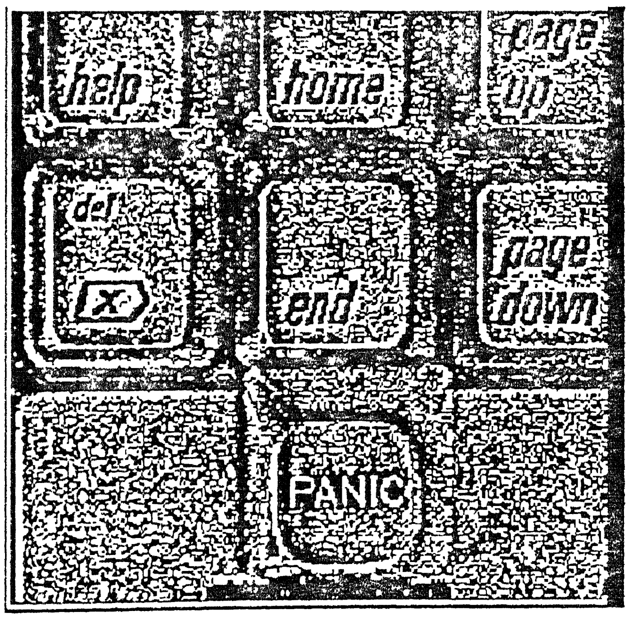


Fig 5.39: Binary image of panic.bmp, using dynamic method.

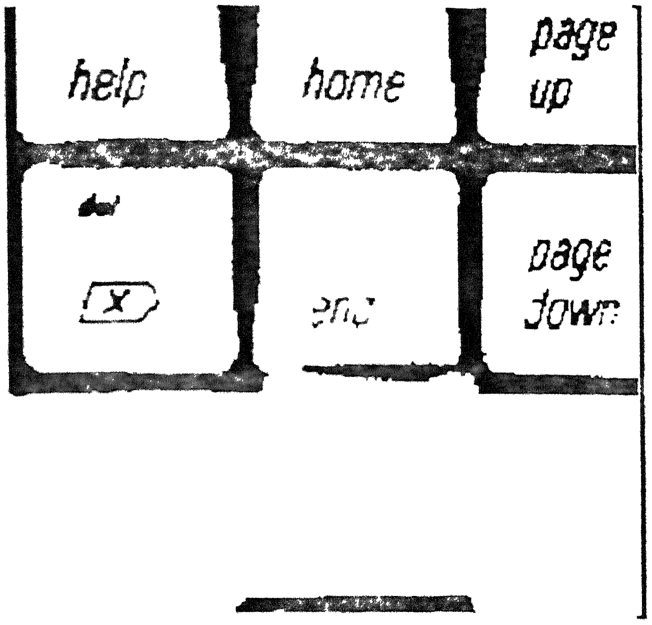


Fig 5.40 : Binary image of panic.bmp, using edge detection method.

Chapter 6

Conclusion

6.1 Summary of work done

This thesis implements part of an image processing work bench and deals with mathematical, logical and statistical operations In addition it also implements: scaling algorithms, rotation and flipping, zoom and dezoom, cut and paste, the clipboard facility - and printing of images The work bench also has a simple batch facility for composing operations In addition the routines are available as DLLs.

The user interacts with the work bench through a graphical point and click interface which is easy to use.

Context sensitive help is also available.

6.2 Suggestions for further Improvements

The work bench evolved in the process of development and features which were not envisaged at the start were added later. However, many more features remain to be added to make the work bench truly useful. Some of these are:

- Provision to read and process standard image formats other than .BMP.
- Grayscale Operations, especially non linear grayscaling operation can be generalized by allowing the user to specify the transformation function. At the moment it has only one transfer function which can enhance the contrast of dark pixels and lower that of bright pixels
- There are various operators that can be used in the thresholding operation, to obtain a corresponding binary image, based on the characteristics of the input image. This package implements just three of those.
- PIF (Program Information Files) facility should be developed further to overcome the present drawback currently observed in MS Windows. Palette operations are

supported in MS DOS, however, the same is not supported in certain adapters when used with MS Windows Taking example of 16 color VGA adapter, which is currently used in most computing facilities, does not allow its palette to be changed when used with MS Windows However, there is no restriction on palette change in MS DOS. Change in palette, allows the system to display colors close to the logical palette desired by the user that helps in yielding very realistic rendering of the images. Not only that standard VGA adapter allows user to program it in its video mode 19 (dec) which gives 256 colors MS Windows using this particular adapter will not allow user to program it from this mode and hence the VGA adapter will not be able to give 256 colors when used from MS Windows. Primary aim of using the PIF facility is to retain the user interface of MS Windows and at the same time using those facilities in MS DOS environment which are currently not supported in MS Windows.

- A general purpose interpreter which allows application programs to be prototyped. Currently, the work bench supports a simple batch facility which allows operators to be composed but does not provide the data structuring or control primitives.

Bibliography

[1] Harley R. Myler and Arthur R. Weeks, *Computer Imaging Recipes in C, PTR* Prentice Hall, 1993

[2] R.C. Gonzalez and P. Wintz, *Digital Image Processing*, Addison - Wesley, 1987.

[3] W.K. Pratt, *Digital Image Processing*, John Wiley & Sons, 1978.

[4] David Vernon, *Machine Vision - Automated Visual Inspection & Robot Vision,* Prentice Hall

[5] Donald Hearn, M. Pauline Baker, *Computer Graphics*, PHI, 1986.

[6] Borland C++ Version 4.0 Reference Guide.

[7] Robert Lafore, *C Programming Using C++*, Waite Group, Inc., 1990.

[8] James L. Conger, *Windows API Bible*, Waite Group Press, 1994.

[9] Charles Petzold, *Programming Windows 3.1*, Microsoft Press, 1992.

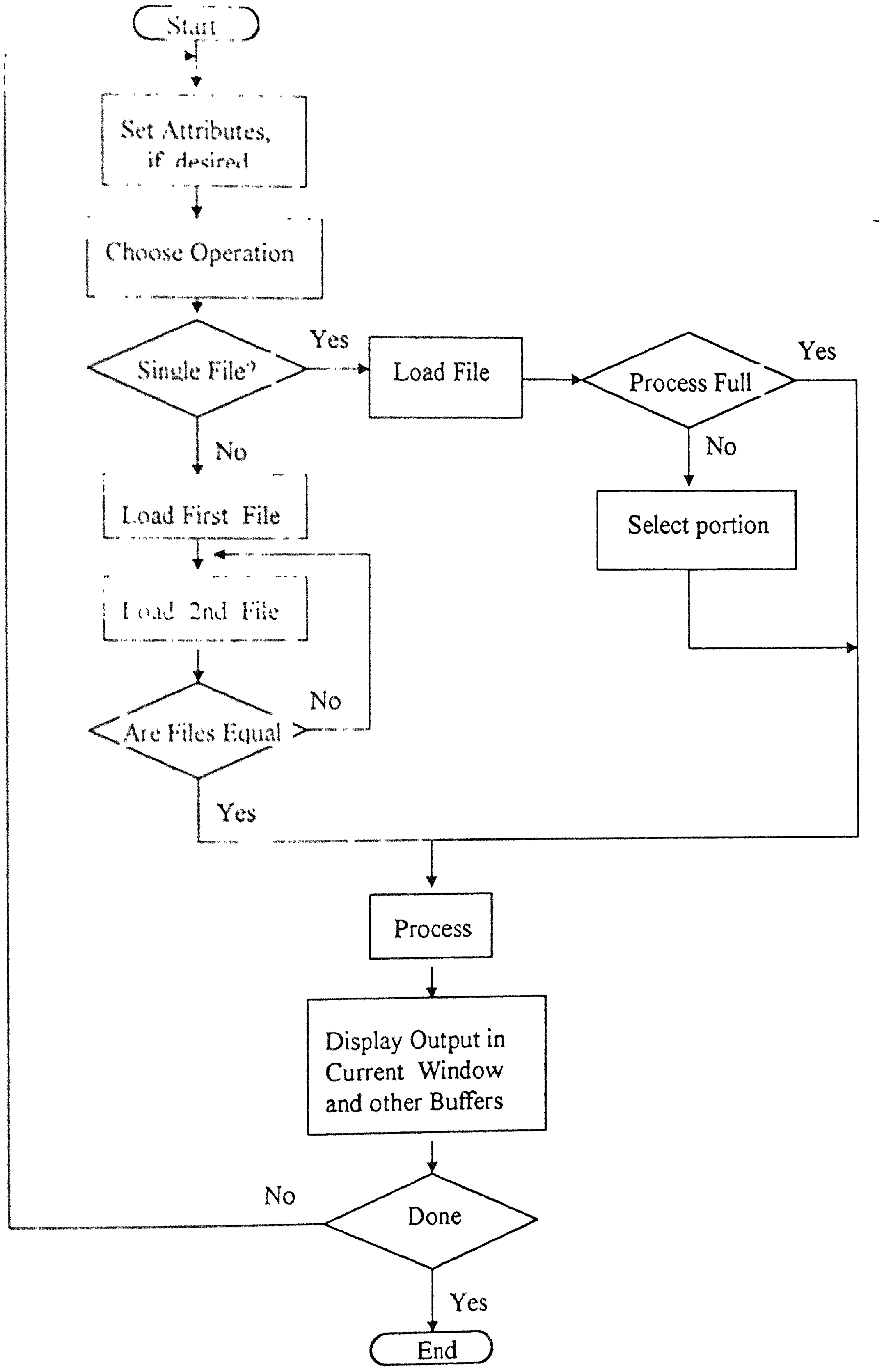
[10] Robert Lafore, *Windows Programming Made Easy*, Waite Group Press, 1993.

[11] Mark Peterson, *Borland C++ Developer's Bible*, Waite Group Press, 1994.

[12] Ben Ezzel, *PC Magazine Windows 3.1 Graphics Programming*, ZIFF-DAVIS PRESS, USA, 1994.

Appendix A

General Sequence of Action



Appendix B

Definitions of certain terms used in text

RLE (Run Length Encoding)	Compresion is used with most image formats to reduce both memory and disk storage requirements. In MS Windows, two formats are supported for bitmap compression, one each for four (BI_RLE4) and eight (BI_RLE8) bits per pixel.
Binary Image	Image having only two levels, white & black.
PIF (Program Information Files)	In order to run DOS programs efficiently, MS Windows require information about how the program uses PC resources including how the program uses internal memory, ports and the monitor. Windows get this information through a PIF.
Pixel Value	Graylevel value (or intensity) of the pixel, in .BMP format, is used as an index into the color table contained inside the BITMAPINFO header.
GDI (Graphical Device Interface)	In MS Windows GDI is frequently used term for collection of functions that provide output to all graphic devices, such as the display screen, the printer, plotters, and so forth. Term "GDI" is also used to refer to the whole approach MS Windows takes to graphics.
GUI (Graphics User Interface)	It is the term used for environments like MS Windows. Its called "graphical" because MS Windows uses graphics to organize the workspace and present the user with intuitive ways to accomplish tasks. MS Windows is called an environment because you can run other programs from within

MS Windows.

Clipboard	MS Windows clipboard feature, is a facility that provides both storage and information transfer between MS Windows applications. A clipboard can handle several types of formats including bitmap formats.
Banding	It is the process of slicing the page into smaller rectangles and printing the pieces one at a time. It helps to avoid constructing the entire page in memory.

Appendix C

BMP File Format

The device independent bitmap (DIB) begins with a file header section defined by **BITMAPFILEHEADER** structure. The structure has five fields:

Field	**Size**	**Description**
bfType	UINT	The type "BM" for Bitmap.
bfSize	DWORD	Total size of the file.
bfReserved1	UINT	Set to 0.
bfReserved2	UINT	Set to 0.
bfOffBits	DWORD	Offset to the bitmap bits from the beginning of the file.

This is followed by another header defined by the **BITMAPINFOHEADER** structure This structure has 11 fields:

Field	**Size**	**Description**
biSize	DWORD	Size of the structure in bytes.
biWidth	LONG	Width of the bitmap in pixels.
biHeight	LONG	Height of the bitmap in pixels.
biPlanes	WORD	Set to 1.
biBitCount	WORD	Color bits per pixel (1, 4. 8

		or 24).
biCompression	DWORD	Compression scheme (0 for none or BI_RGB).
biSizeImage	DWORD	Size of bitmap in bytes.
biXPelsPerMeter	LONG	Horizontal resolution in pixels per meter.
biYPelsPerMeter	LONG	Vertical resolution in pixels per meter.
biClrUsed	DWORD	Number of colors used in image.
biClrImportant	DWORD	Number of important colors in image.

All fields following the biBitCount field may be set to 0 for default values (or may not even appear in the file). Thus, the structure can be as small as 16 bytes in length. It can also contain additional fields beyond those shown here.

If biClrUsed is set to 0 and the number of color bits per pixel is 1, 4, or 8, the **BITMAPINFOHEADER** is followed by a color table, which consist of 2 or more **RGBQUAD** structures. The **RGBQUAD** structure defines an RGB color value.

Field	**Size**	**Description**
rgbBlue	BYTE	Blue Intensity.
rgbGreen	BYTE	Green Intensity.
rgbRed	BYTE	Red Intensity.

rgbReserved	BYTE	Set to 0.

The number of RGBQUAD structures is usually determined by the biBitCount field Two RGBQUAD structures are required for 1 color bit, 16 for 4 color bits, and 256 for 8 color bits However, if the biClrUsed field is nonzero, then the biClrUsed field contains the number of RGBQUAD structures in the color.

The color table is followed by the array of bits that define the bitmap image. This array begins with the bottom row of pixels. Each row begins with the leftmost pixels. Each pixel correspond to 1, 4, 8, or 24 bits.

For a monochrome bitmap with 1 color bit per pixel, the first pixel in each row is represented by the most significant bit of the first byte in each row. If this bit is 0, the color of the pixel can be obtained from the first RGBQUAD structure in the color table. If the bit is 1, the color is given by the second RGBQUAD structure in the color table.

For a 16 color bitmap with 4 bits per pixel, the first pixel in each row is represented by the most significant 4 bits of the first byte in each row. The color of each pixel is obtained by using the 4 bit value as an index into the 16 entries in the color table.

For a 256 color bitmap, each byte corresponds to one pixel. The color of the pixel is obtained by using the 8-bit value as an index into the 256 entries in the color table.

If the bitmap contains 24 color bits per pixel, each set of three bytes is an RGB value of the pixel. There is no color table (unless the biClrUsed field in the BITMAPINFOHEADER structure is non zero).

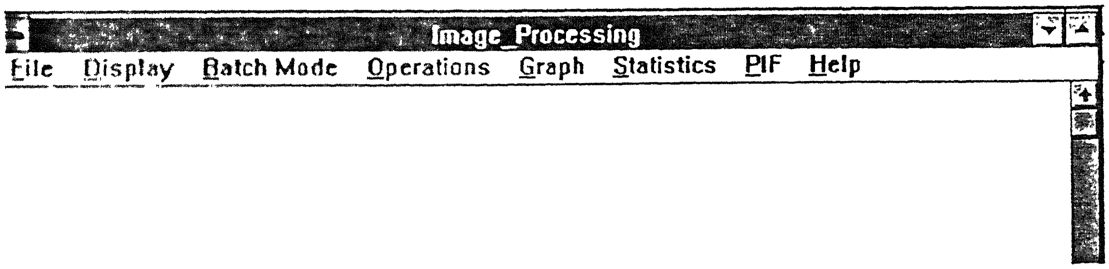
In each case, each row of the bitmap data contains a multiple of 4 bytes. The row is padded to the right to ensure this.

Appendix D

User Manual

User Manual gives a brief glimpse of the various menu items and dialog boxes that are encountered while using this package.

This package being a Windows application can be run as any other Windows application either from the File manager or Program manager. Fig D.1 displays the opening (main) menu as the package is being run.



Image_Processing
File Display Batch Mode Operations Graph Statistics PIF Help

Fig D.1: Main Program Menu

Before doing any operation it is best to set attributes by selecting **File | Attributes** menu item. A pull down menu appears as shown at Fig D.2 for specifying the various attributes. Further by selecting the Pen attribute a dialog box as shown at Fig D.3 appears.

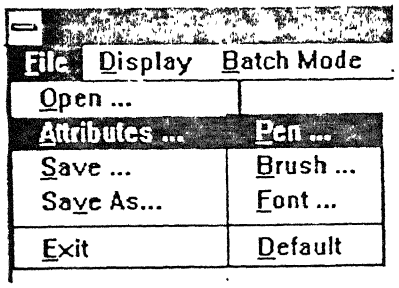
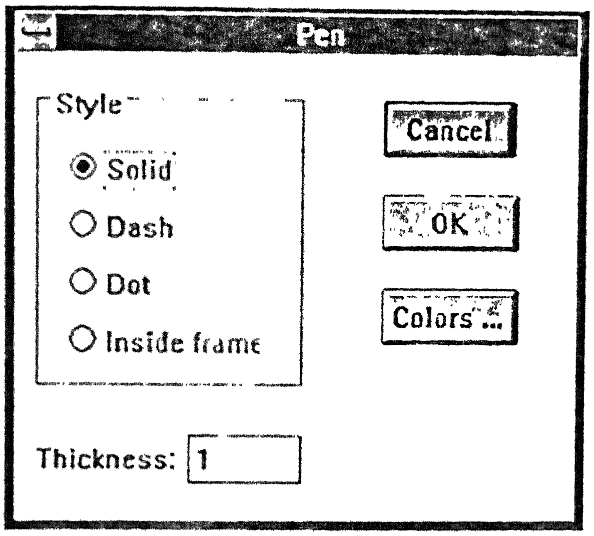


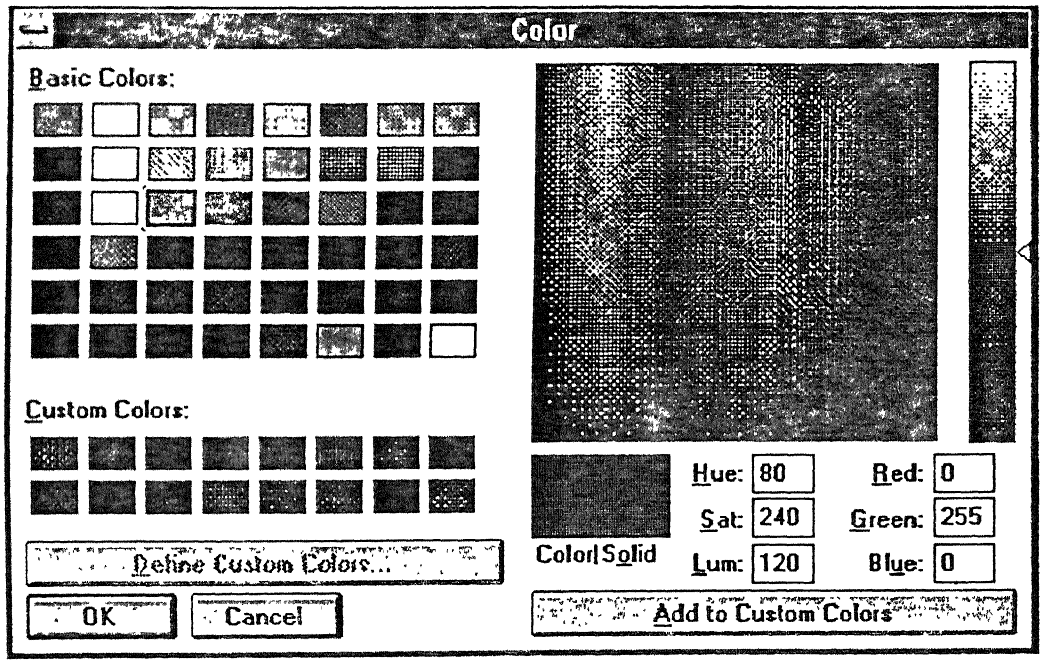
Fig D.2: Attributes Menu.



Thickness: 1

The pens width and style can be specified here. To choose color of the pen click **color** push button and another dialog box for choosing the custom color appears as shown at Fig D.4. User can specify the blend of color and

Fig D.3: Dialog box for choosing Pen



Hue: 80
Sat: 240
Lum: 120
Red: 0
Green: 255
Blue: 0

Fig D.4: Dialog box for choosing custom color

press **OK** to return to the pen dialog box. Once done with it user can press **OK** to return back to the pull down menu. Likewise an user can press **File | Attributes | Brush** menu item to choose the brush attribute.

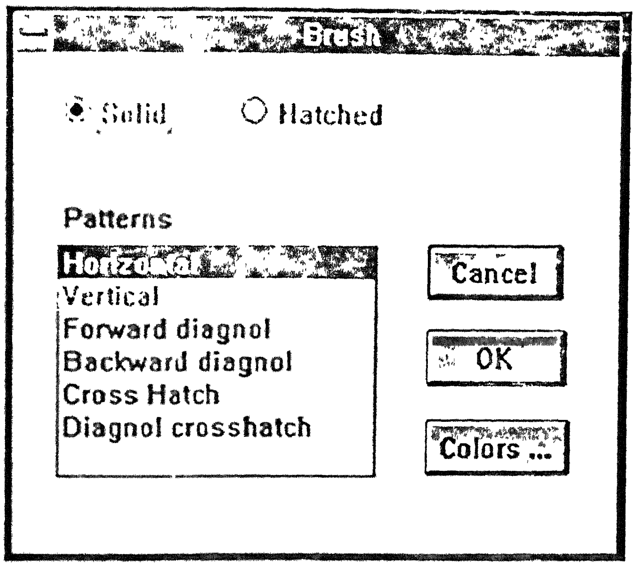


Fig D.5: Dialogbox for choosing Brush

A Dialog box at Fig D.5 illustrates the situation. A user can choose either a solid brush or a hatch brush. If hatch brush is chosen then one of the brush styles in the list box can be chosen. For giving color to the brush color push button can be pressed. Color dialog box shown at Fig D.4 appears and color can be chosen just as for the pen. Fig D.6 shows the Font selection dialog box . The user can choose the desired font, its style,

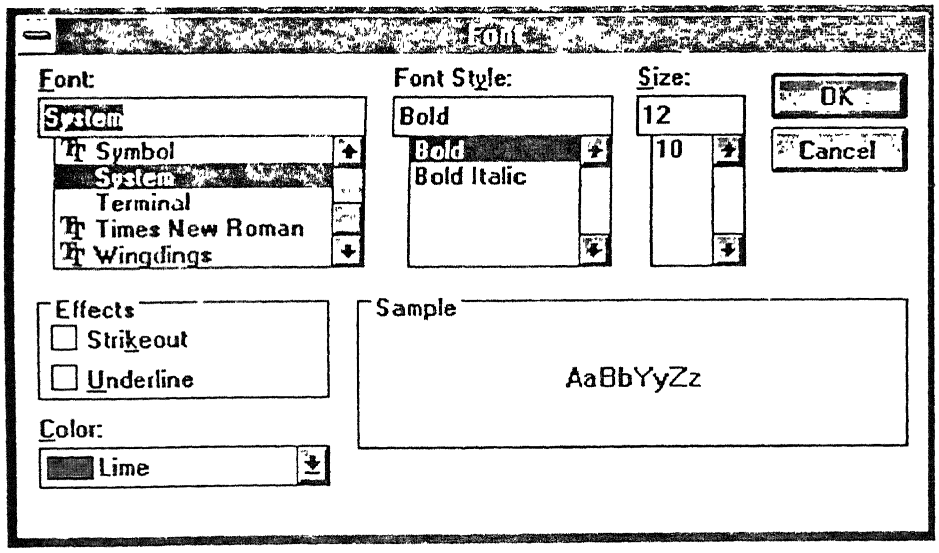


Fig D.6: Dialogbox for choosing Font

size, color etc.

The sample text in the dialog box shows the changes effected when the user selects various options. As mentioned, the font chosen by the user comes in handy while labeling the various objects on the screen. **File | Attributes | Default** menu item restores the default system font which has a black color. Choosing **File | Open** menu item from the pull down menu brings up the dialog box shown at Fig D.7.

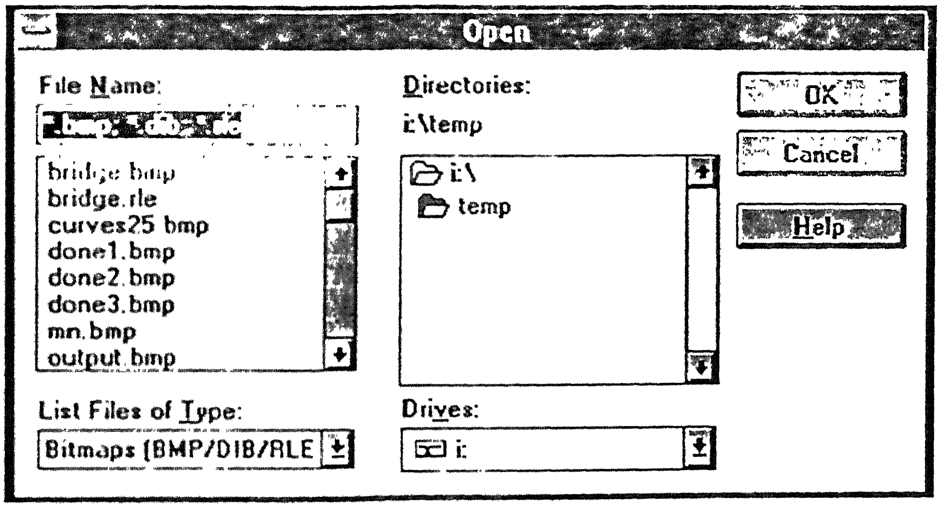


Fig D.7: File Open Dialogbox

This dialog box is used for loading files. The user can either type in the name of the file to be viewed/processed or pick up the file name from the list box displayed as part of the **File Open** dialog box. The dialog box enables a user to navigate the disk in pursuit of a particular file, which can be in a different drive or sub directory. **File | Save | Full** menu item may be used to save the original or processed file. By default, if the image buffer is not empty then the processed image is saved under the filename *result.bmp* and original filename as *orig.bmp*. On successful completion of the saving operation suitable prompts are given. By choosing **File | Save | Partial** option, only the image portion selected by the mouse is saved. **File | Save As | Full** and **File | Save As | Partial** menu items are similar to the last two menu items. But they also display a **File Save As** dialog box shown at Fig D.8 thus allowing a user to give the file name.

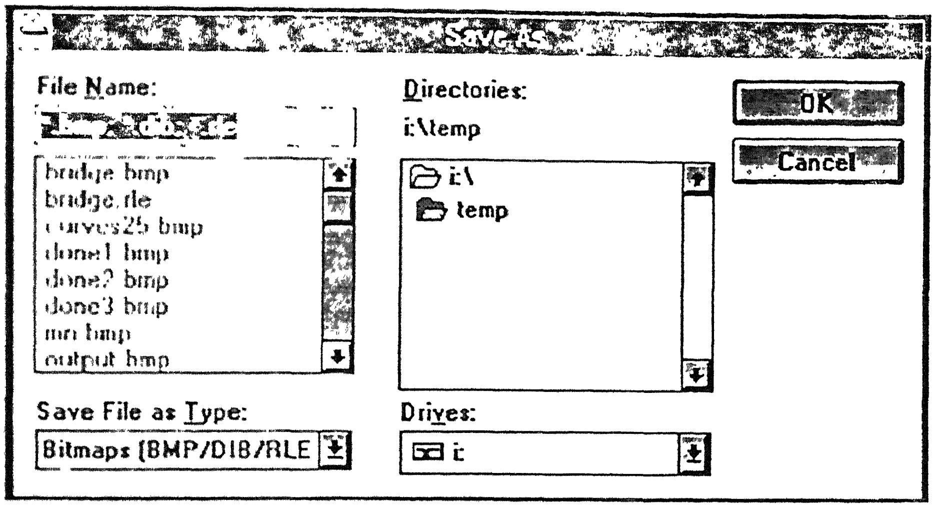


Fig D.8: File Save Dialogbox

The operation of this dialog box is similar to the File Open dialog box. In case the user enters a file name that already exists on disk, an overwrite prompt is displayed to warn the user. **File** | **Exit** menu item is chosen when the user no longer wants to use the work bench and desires that the application be terminated. After processing an image, the user can revert to this menu to have a look at the original file by choosing **Display** | **Original** menu item. By choosing **Display** | **Processed** menu item user can view the processed file maintained separately in a buffer. Nevertheless, the processed image is shown, by default in the current window along with relevant plots after performing each operation. Fig D 9 shows the pull down menu for display menu item.

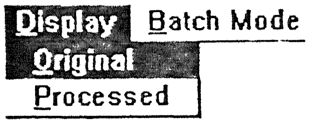


Fig D.9: Display Menuitem

Selecting the **Batch Mode** menu item brings up the batch mode menu as shown at Fig D.10. The **File** menu for batch mode brings down the pull down menu shown at Fig D.11. **File** | **Load** menu enables a user to load an image file for batch mode operation, File Open dialog box shown at Fig D.7 appears.

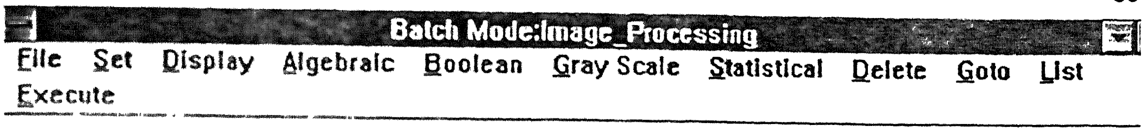

Batch Mode:Image_Processing
File Set Display Algebraic Boolean Gray Scale Statistical Delete Goto List Execute

Fig D.10: Main menu for Batch Mode

At this stage the user will be suitably prompted if he desires to work with single image or two images.

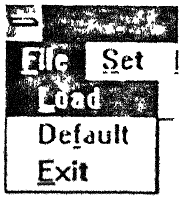
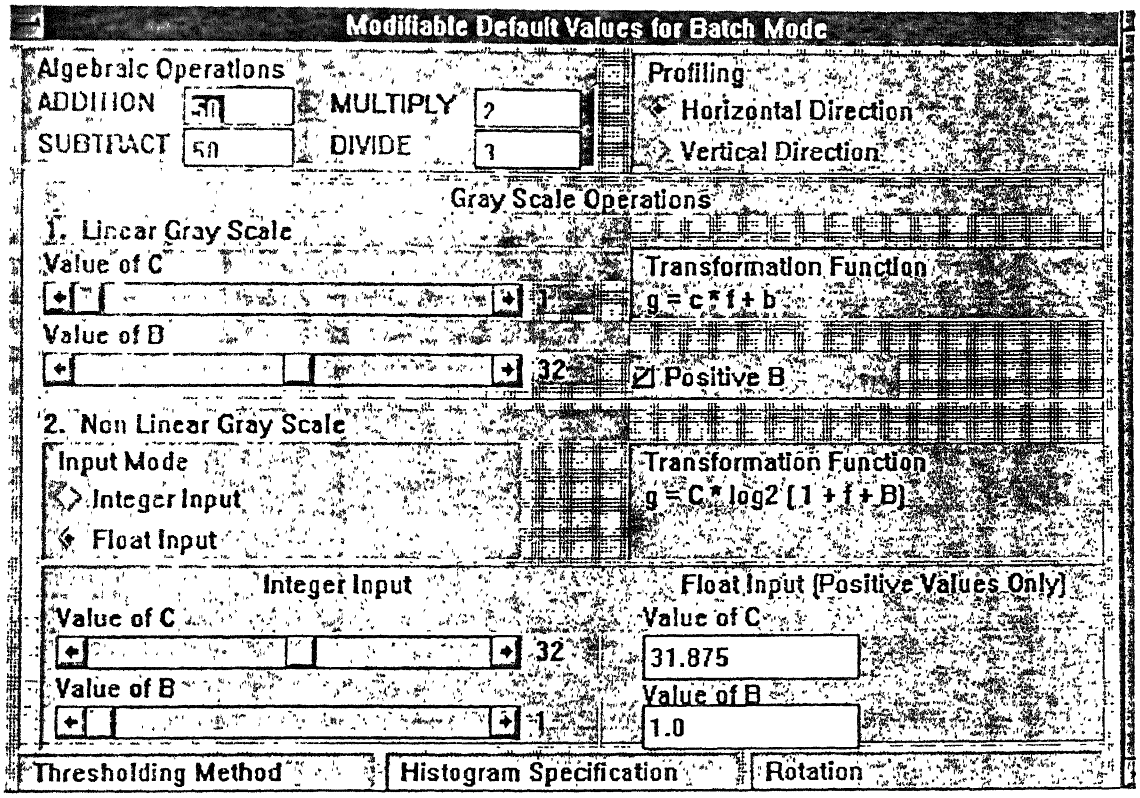


Fig D.11: File menu in Batch Mode

Accordingly, if the user decides to work with a single image he has an option to process the whole file or part of it. If the user decides to process part of the file then he/she must select the portion desired to be processed using mouse. The **File | Default** menu item brings up **Batch Mode Default Modifiable Values** dialog box as shown at Fig D.12. User can view default settings for various operations. These settings may be accepted as it is or modified by the user. These settings apply to those operations that are listed in the batch mode list constructed by the user. This dialog box can be asked by the user any time, during the batch mode session, for viewing/modification. The **File | Exit** menu item exits from the batch mode and control returns to the main program with the caption and menu of the main program restored as shown at Fig D.1. The **Set | Reset** menu item enables a user to build a fresh list of commands for batch mode operation after finishing the last batch mode session. The **Set | Redo** menu item enables a user apart from building a fresh list of commands, to load new input image/images for further batch mode operation. Fig D.13 shows the drop down menu that is seen if the **Set** menu item is clicked. Figures D.14, D.15, D.16 and D.17

shows the drop down menus for Algebraic, Boolean, Statistical, Grayscale and Thresholding operations.



Modifiable Default Values for Batch Mode

Algebraic Operations
ADDITION 30
SUBTRACT 50
MULTIPLY 2
DIVIDE 1

Profiling
Horizontal Direction
Vertical Direction

Gray Scale Operations

1. Linear Gray Scale
Value of C 1
Value of B 32
Transformation Function
g = c * f + b
☑ Positive B

2. Non Linear Gray Scale
Input Mode
Integer Input
Float Input
Transformation Function
g = C * log2 [1 + f + B]

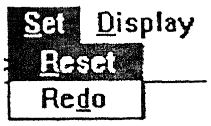
Integer Input
Value of C 32
Value of B 1

Float Input (Positive Values Only)
Value of C 31.875
Value of B 1.0

Thresholding Method | Histogram Specification | Rotation

Fig D.12: Batch Mode Default Values Dialogbox

As soon as the file is successfully loaded it is marked by the appearance of an empty list box shown at Fig D.18.



The user can select operations to be performed from the appropriate pull down menu of operations.

Fig D.13: Set Menu in Batch Mode

For example, Fig D.19 shows the list box after a few commands have been chosen by the user. The user can delete items from the batch list at this stage by choosing the **Delete** menu item. This deletion may not be possible after **Execute** command is given. The **Execute** menu item is an important command that must be issued by the user before execution of commands listed in the list box. After each item/operation in the list box is executed, the user can watch the various outputs generated by clicking the **Display** menu item which brings down a pull down menu shown at Fig D.20. Although by default, the processed output is displayed in the current window as soon as any operation is over the user can view additional outputs, as applicable to operations, by clicking sub items in the pull down menu. After having done with the last operation, the user can click the **Goto** menu item to execute next command in the list box. The process of viewing the last output and **Goto** menu item continues until the last command/item in the list box is executed. Fig D.10 displays

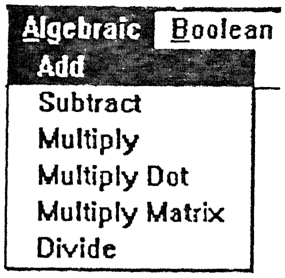


Fig D.14: Algebraic menu in Batch Mode.

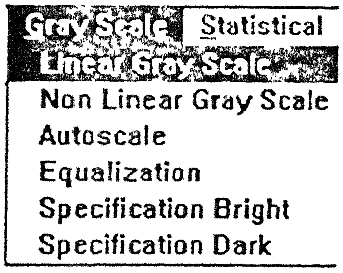


Fig D.16: Grayscale menu in Batch Mode

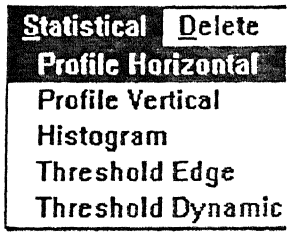


Fig D.17: Statistical & Thresholding menu items in Batch Mode.

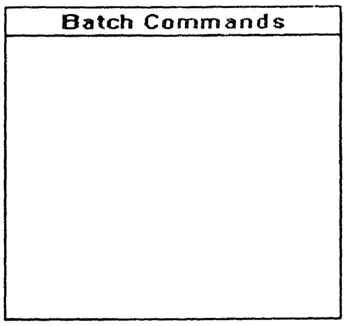


Fig D.18: Empty List Box in Batch Mode.

the **Delete**, **Goto** and **Execute** menu items. At times during the batch mode session the user would like to hide the list box which may be unnecessarily cluttering the desktop. In that case the user can resort to menu item **List** to achieve this.

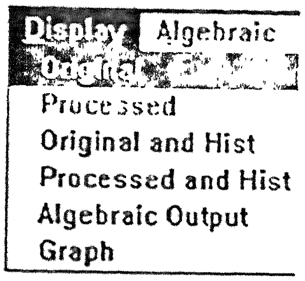


Fig D.20: Display Menu item in Batch Mode.

List | Hide sub menu item will hide list box and **List | Display** sub menu item is used for achieving the reverse. Fig D.21 shows the menu item **List** with its pull down menu.

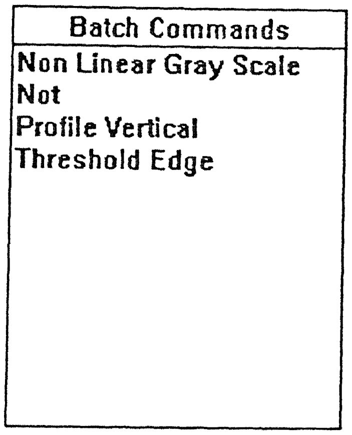


Fig D.19: List Box with few commands.

Selecting the **Operations** menu item in the main program menu brings a drop down menu as shown at Fig D.22.

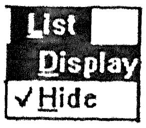


Fig D.21: List Menuitem in Batch Mode.

Further, selecting the **Boolean** menu item brings down another pull down menu shown at Fig D 23. Boolean operations except for Boolean NOT require two input images for processing.

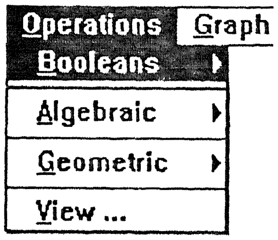


Fig D.22: Operations Menuitem.

On clicking any Boolean operation File Open dialog box shown at Fig D.7 appears. In case of two input image operations this dialog box is presented twice. After the image is loaded it is displayed in the window. For two

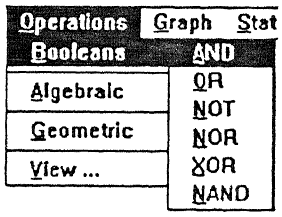


Fig D.23: Booleans Menuitem.

input operations, it is important that two input images should of equal sizes. After processing, the output is displayed along with original input files in the current window. The user can also find and view original and processed files separately as mentioned by choosing **Display** | **Original** and **Display** | **Processed** menu items respectively. Selecting **Operations** | **Algebraic** menu item brings down a pull down menu as shown at Fig D.24.

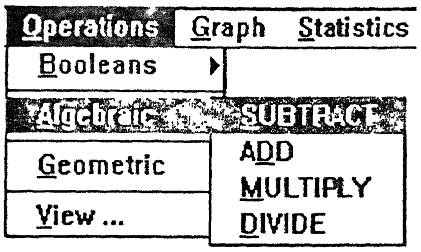


Fig D.24: Algebraic Menuitem.

Algebraic operations allow both single and double input image operations. For single input image operation, after loading the image a dialog box shown at Fig D.25 appears. User can enter any positive value between 0 and 255 in the edit box. It is suggested that the value be as low as possible, more so in case of multiply and divide operations.

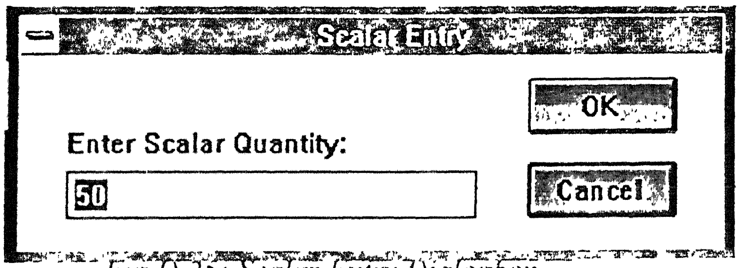


Fig D.25: Scalar Entry Dialogbox.

The reason for this is that if the values entered are higher then intermediate results become quite large and have to be ultimately scaled down to an unsigned char value, for displaying, which usually results in loss of accuracy. This precaution will yield comparatively accurate results. In case of two input images the relevant operation is performed between two images, before results are displayed in the window. The user can view the original and processed images separately by selecting respective sub menuitem of **Display** menuitem. Fig D.26 shows

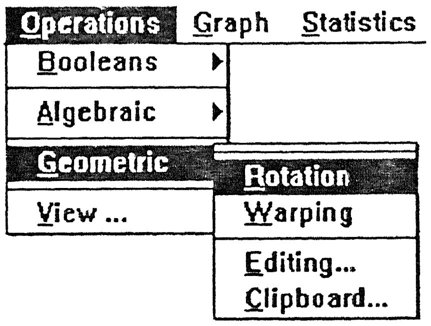


Fig D.26: Geometric Menuitem.

the pull down menu that appears once the Geometric sub menuitem is clicked. Selecting **Operations | Geometric | Rotation** sub menuitem performs the rotation by a specified angle and direction desired (clockwise or anti clockwise), on an input image. The dialog box shown at Fig D.27 appears and it enables the user to enter the specified angle and the direction of rotation for the output image.

Selecting **Operations | Geometric | Editing** sub menuitem brings up a popup child window which is shown at Fig D.28 along with its menu. Selecting Shapes menu item from the child window brings down a pull down menu as shown at Fig D.29.

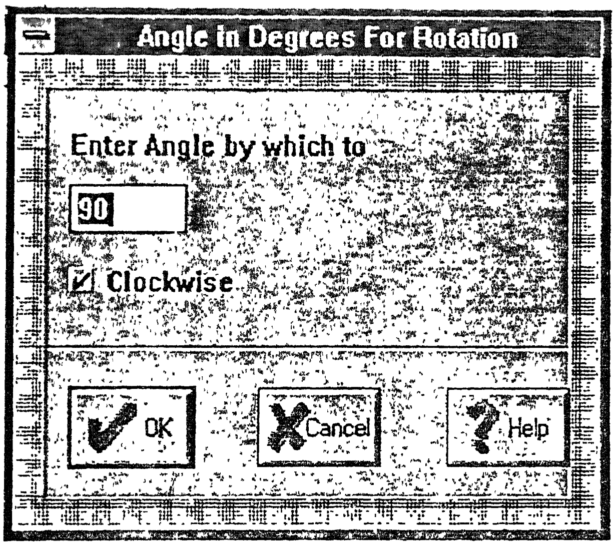


Fig D.27: Angle Entry Dialogbox for Rotation.

Pull down menu for **Commands, Brushes, Pens, Background** and **ROP** codes are shown at Figures D.30, D.31, D.32, D.33 and D.34 respectively. The user can select any

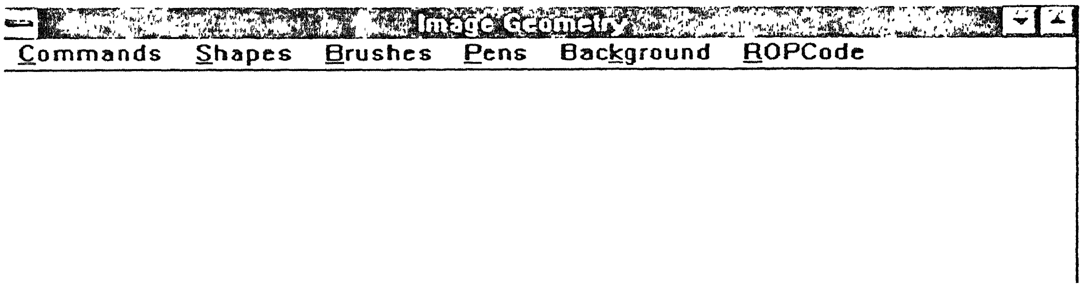


Fig D.28: Main menu for Editing menuitem.

shape item from the **Shapes** menu item. These items can be drawn in the window using the mouse. The current settings for Brushes, Pens , Background and ROP code will be effective, which can be varied to see their effects when the shapes are redrawn.

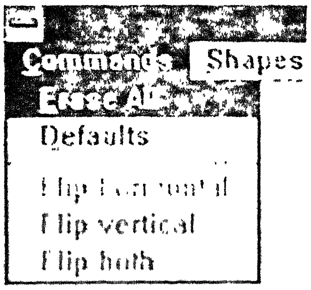


Fig D.30: Commands Menu.

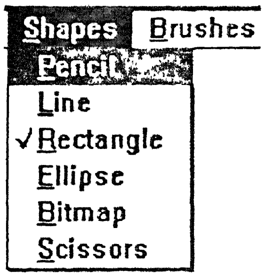


Fig D.29: Shapes Menu.

Of particular interest are the **Shapes | Bitmap** menu item, which enables a user to load a bitmap image of choice before doing any geometric operations with it. The user can select the menuitem and mark the portion to be selected on the bitmap or any other shape drawn for that matter, by using the mouse.

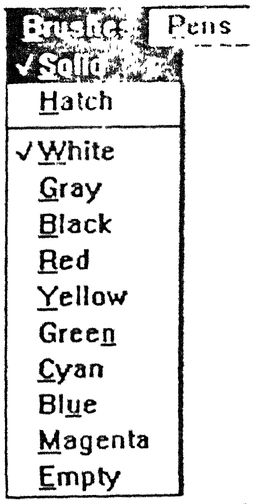


Fig D.31: Brush Menu.

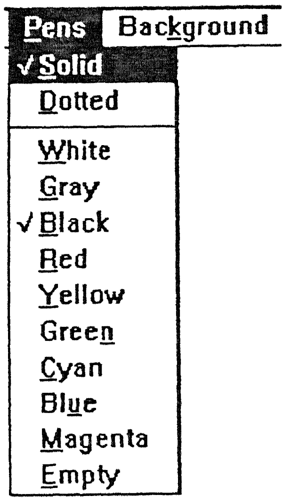


Fig D.32: Pen Menu.

Once a portion is selected, marked by a dotted rectangle, the user can then take the mouse pointer anywhere within the selected region, press the left mouse button and drag the mouse pointer while the left mouse button is pressed. The user will notice as the mouse pointer moves the selected portion of the image is cut from the original image and follows the mouse pointer. This cutpiece of the image/figure can be placed anywhere on the screen by releasing the left mouse button at that location. Selected portions of the image can also be copied from the original image and placed at a new location, instead of being cut from the source, by pressing the **Cntrl** (control) key and taking similar actions as just done for cutting the image.

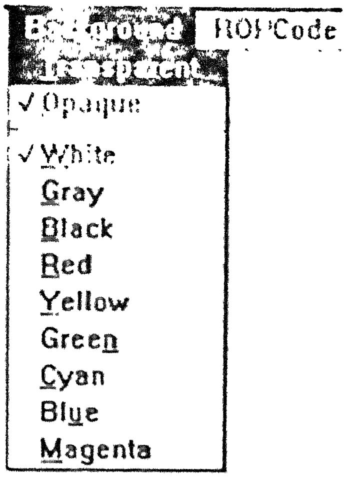


Fig D.33: Background Menu.

Menuitems **Commands | Flip horizontal** flips the selected portion of the image horizontally. Likewise, menuitem **Commands | Flip Vertical** flips the selected portion of the image vertically.Menuitem **Commands | Flip both** can be used to flip horizontally as well as vertically simultaneously. If the **shift** key is pressed then the outlined area can be dragged further to be expanded and the result is we get a zoomed picture. **Commands | Erase All** menuitem erases everything on the desktop and menuitem **Commands | Defaults** restores the default shape item, brush, pen, background and ROP code.

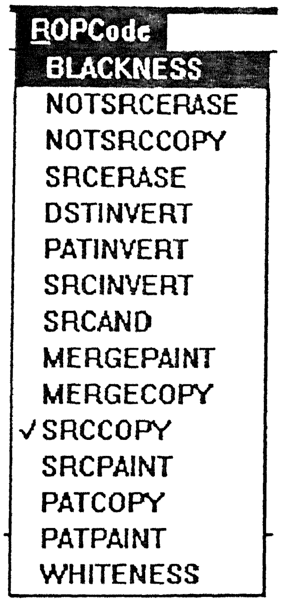


Fig D.34: ROP Code menu.

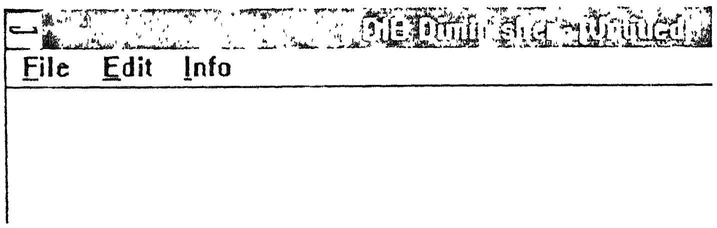


Fig D.35: Popup menu for Clipboard.

Fig D.35 shows the menuitem that accompanies the popup window that comes as soon as the sub menuitem **Operations | Geometric | Clipboard** is clicked. Clipboard is a Windows way of exchanging data with its applications. Fig D.36 shows the drop down menu that appears as the File menuitem is selected from its menu.

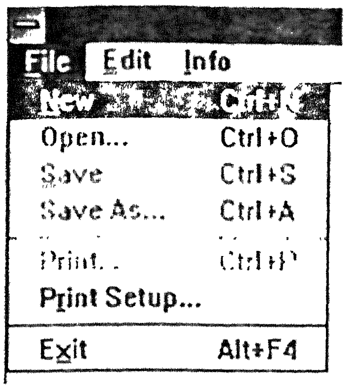


Fig D.36: Dropdown Menu for File.

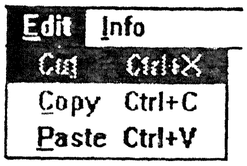


Fig D.37: Dropdown Menu for Edit.

Menuitems **File | New, File | Open, File | Save, File | Save As** have their usual meanings. **File | Print** and **File | Print Setup** menuitems comes in handy to print a DIB file. **File | Exit** menuitem allows a user to exit from the current popup child window and return to the main program window. Fig D.37 shows the pull down menu that appears when the **Edit** menuitem is clicked. Menuitem **Edit | Cut** puts the bitmap image from the program to the Windows clipboard. **Edit | Copy** menuitem transfers just a copy of it. **Edit | Paste** copies the clipboard bitmap image into the program, which can immediately be seen on the screen. Of particular utility is the sub menuitem **Info | Measurements** which gives all the details of a bitmap image file. Fig D.38 shows the pulldown menu for it. On pressing it a dialogbox opens up dealt with separately in *Results, Chapter 5*. The user can convert an uncompressed bitmap image to RLE format and viceversa by pressing the **RLE format** and **RGB format** pushbuttons respectively. Conversion is carried out in real time and updated details of the bitmap image can be immediately seen in the dialog box. Infact images in the uncompressed (.BMP) or uncompressed (.RLE) states can be saved to disk or read from it. Major advantage of the compressed images accrues from the fact that it saves a lot of disk space in that state. Menuitem **Operations | View** comes in handy to review the default program output again for Boolean and Arithmetic Operations after the user has clicked menuitems **Display |**

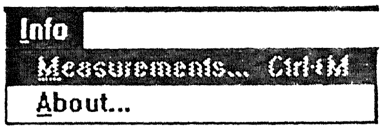


Fig D.38: Dropdown menu for Info.

Original or **Display | Processed** to view the original or processed image respectively. Fig D 39 shows the pull down menu that appears when the **Statistics** menuitem is clicked.

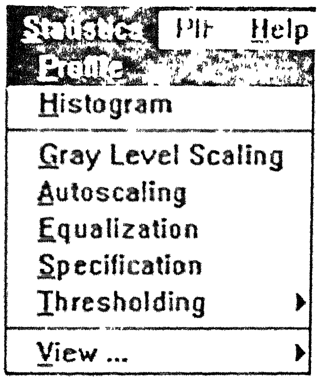


Fig D.39: Pulldown Menu for Statistics.

Menuitem **Statistics | Profile** enables a user to perform profile analysis. User can mark the area to be profiled by using the mouse, as explained under *section 3.2, Selection by Mouse*. The program prompts the user if he/she desires to analyze in the **horizontal** or **vertical** direction. Fig D.40 displays the dialog box which takes this direction. Two radio buttons are displayed one for each direction, the user can click the button of his choice. Programs output is displayed in the window and further allows for extensive viewing by showing portions of the screen desired by the user which is made possible by the **Profile Analysis dialog box** shown at Fig D.41. This dialog box continues to appear at periodic intervals till such time the user decides otherwise and presses the CANCEL pushbutton. **Statistics | Histogram** sub menuitem enables a user to compute the histogram of a bitmap image file. Default output, a pictorial representation of the histogram is displayed in the current window.

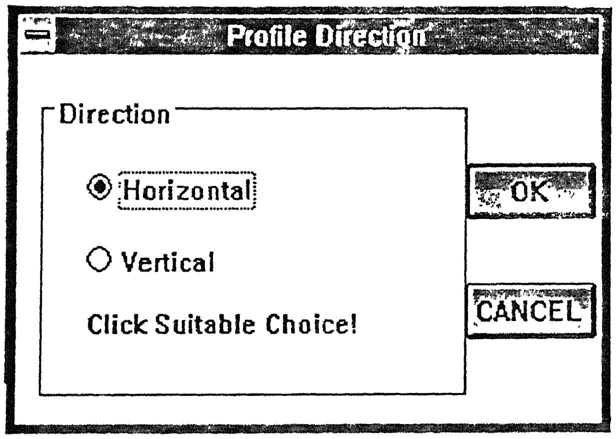


Fig D.40: Profile dialogbox for Horizontal or Vertical direction.

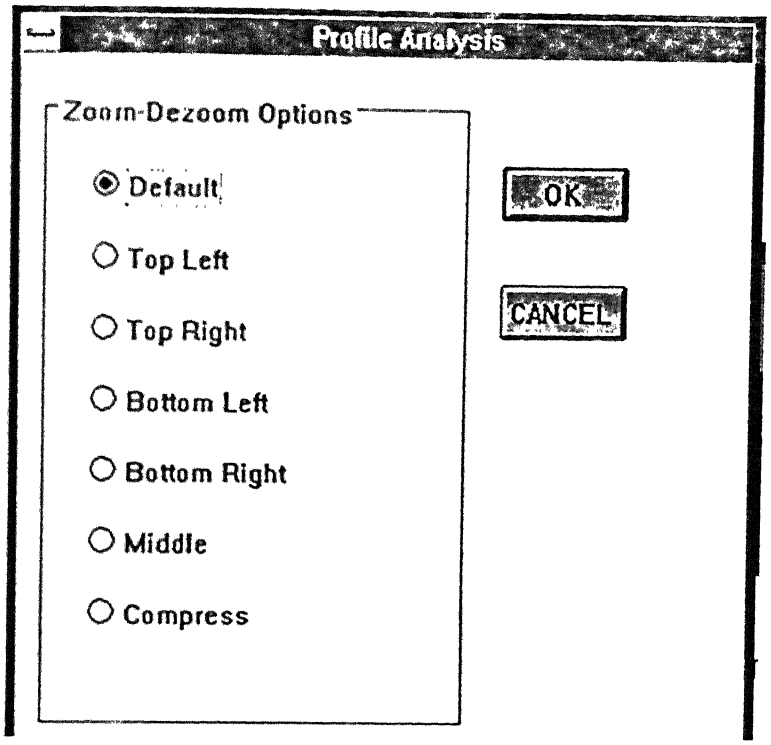


Fig D.41: Dialogbox for Profile Analysis.

For both these operations (profiling and histogram) user can find the default output by clicking the **Graph** menuitem, which will perhaps be needed once the user decides to review it after having a look at the original image from **Display | Original** menuitem. Fig D.42 shows the dialog box that appears when the user clicks the **Statistics | Graylevel** Scaling sub menuitem, after a file has been loaded. The user has option of selecting **Linear** or **Non linear Graylevel** scaling available through the two radiobutton controls. In either case further input dialog boxes appear for the two shown at Figures D.43 and D.44.

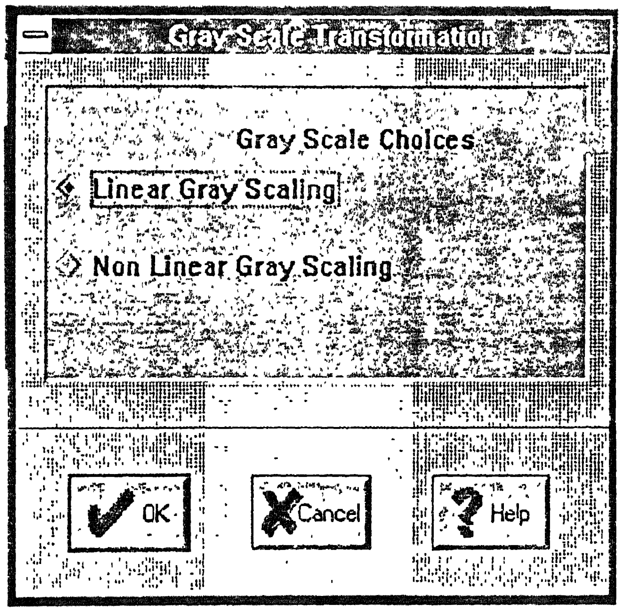
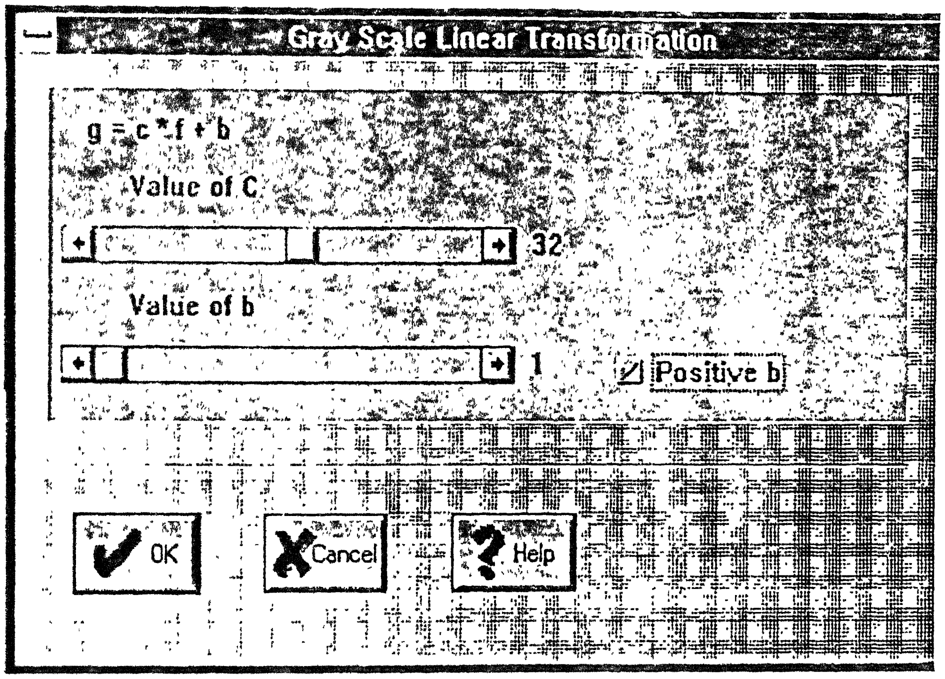


Fig D.42: Dialogbox for choosing Linear or Non linear graylevel.



g = c * f + b

Value of C

32

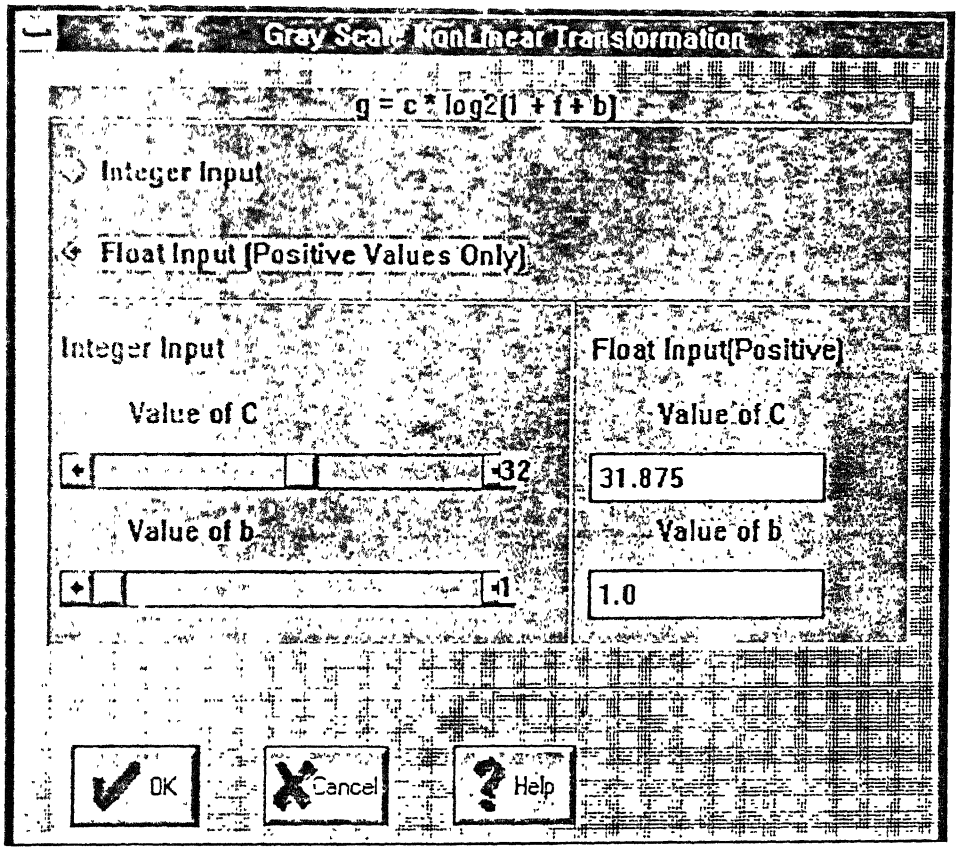
Value of b

1

☑ Positive b

Fig D.43: Input dialogbox for Linear grayscale.

As is customary in all dialog boxes, they also display default settings. Scrollbar controls in both the input dialog boxes can be used to enter integer input. Checkbox control in the Linear input dialog box can be used to indicate positive or negative values for the parameter **b**. In the Non linear input dialog box the user has the option to indicate **integer** or **float** input. Integer input is entered through scrollbar controls as in linear input. In case the user desires to enter float values, then values for two parameters **b** and **c** have to be entered at the places indicated in the dialog box. These values should be positive to have any meaningful results. The program produces as its output the grayscaled image and its transformation plot in the current window. For other outputs pertaining to this operation the user can see **Display | Original, Display | Processed** and **Statistics | View | Original & Hist** sub menuitems. In particular the last sub menuitem also gives the pictorial plot of the histogram of the input image. Fig D.45 shows the dialog box that appears when the user clicks the **Statistics | Autoscale** sub menuitem.


Gray Scale NonLinear Transformation

g = c * log2(1 + f + b)

Integer Input

Float Input [Positive Values Only]

Integer Input

Value of C

32

Value of b

1

Float Input[Positive]

Value of C

31.875

Value of b

1.0

Fig D.44: Input dialogbox for Non linear grayscaling.

Assuming that the image has been loaded, the user can indicate the lowest and highest graylevel he desires in the output image. The necessary inputs can be made through the scrollbars. Program output appears in the current window which will be the output image with graylevel range specified by the user along with its histogram. Other relevant output for the operation can be found at similar places as for grayscale operations. Selecting **Statistics | Equalization** sub menuitem enables a user to perform the Equalization on the input image. Output for the program is displayed in the current window. Other outputs of the operation can be found at places similar to the last two operations. Choosing **Statistics | Specification** sub menuitem enables a user to get the output image based on his/her histogram (called desired histogram).

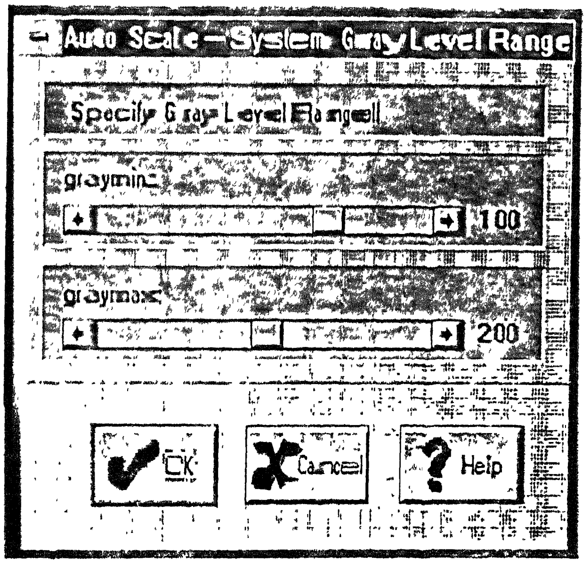


Fig D.45:- Input dialogbox for Autoscaling.

Fig D.46 shows the dialog box that pops up to take the user input. Click on the bright radio button will yield a brighter output and a similar click on the dark radio button will yield comparatively a darker output. If the user prefers the third choice by clicking the user-defined radio button, then the edit control to the right of it becomes active and the user is expected to enter 256 values for the desired

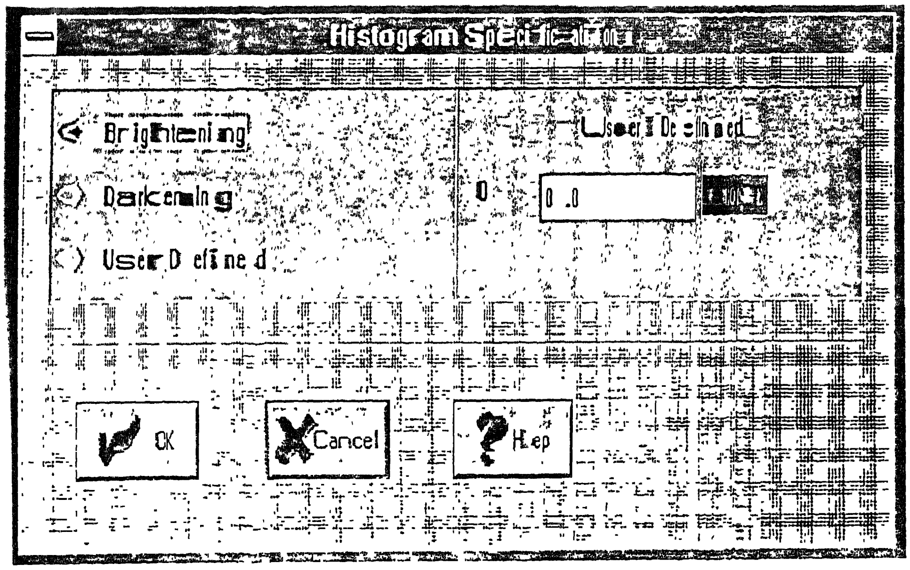


Fig D.46: Dialogbox for choosing Specification Method

histogram. In case the user prematurely aborts the data entry process in that case the program yields the output based on the specified bright histogram. Output for the program will be displayed in the current window. Menuitem Statistics | Thresholding | User Defined enables a user to calculate threshold value and obtain a binary image based on this threshold value The program computes the histogram of the image and displays it waiting for further user action. The user is expected to locate a suitable threshold value from the image histogram, by clicking the right mouse button. Program displays the output in the current window which will be a binary image and its histogram. Menuitems **Statistics | Thresholding | By Edge Detection** and **Statistics | Thresholding | Dynamically** are other two ways to threshold an image. The display process of all thresholding operations is similar to those for grayscaling operations. Fig D.47 shows the pull down menu that appears when the sub menuitem Thresholding is clicked.

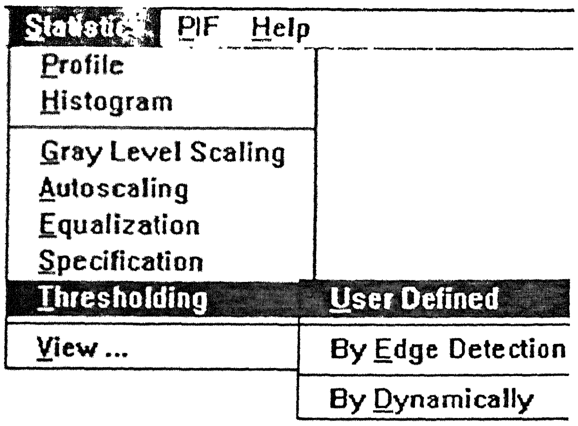


Fig D.47: Pulldown menu for Thresholding.

Menuitem **PIF** is merely a place holder for DOS based applications and can be used to give the desired functionality to this package which cannot be extended through MS Windows. Using PIF (Program Information Files) it is possible to run purely DOS based programs from MS Windows. In other words, it is possible to retain advantages of both platforms namely - Windows GUI and direct screen handling features permitted in DOS such as palette change of video hardware which is currently not permitted in MS Windows for a 16 color VGA adapter. This handicap in MS Windows has resulted in unfaithful display of graylevel images having more than 16 graylevels. This option needs to be further developed.

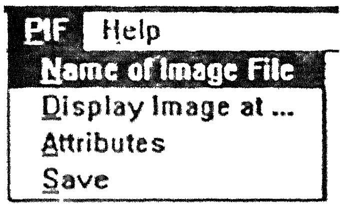
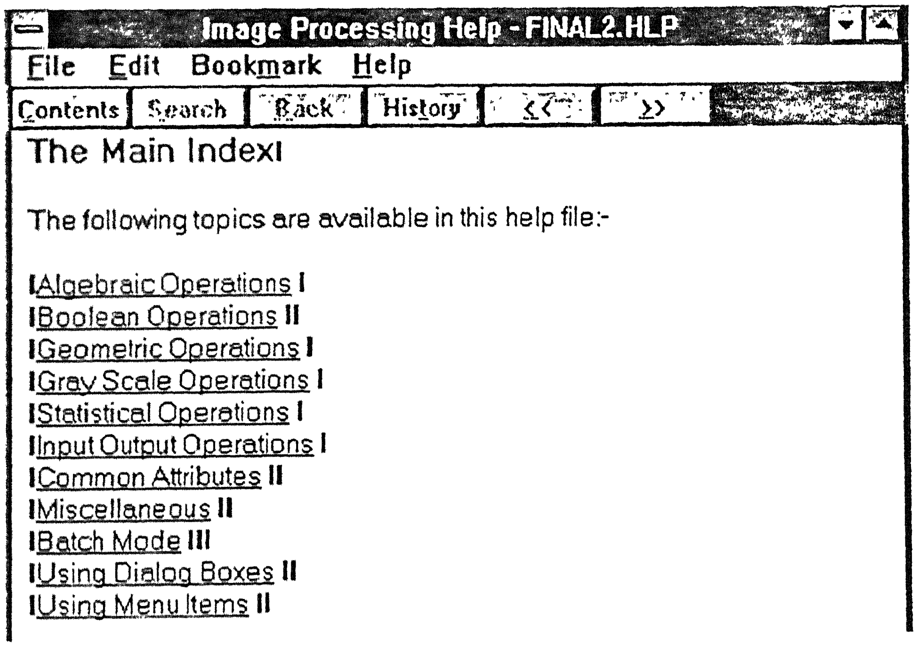


Fig D.48: PIF Menu.

Fig D.48 shows the pull down menu for PIF and can serve as the starting point for further work. Menuitem **Help** gives the access to the Help system. Fig D.49 shows the Help Index that appears on clicking the Help menuitem.



The Main IndexI

The following topics are available in this help file:-

IAlgebraic Operations I
IBoolean Operations II
IGeometric Operations I
IGray Scale Operations I
IStatistical Operations I
IInput Output Operations I
ICommon Attributes II
IMiscellaneous II
IBatch Mode III
IUsing Dialog Boxes II
IUsing Menu Items II

Fig D.49: Help System Main Index.